CHAPTER

4

UNDERSTANDING AND MANAGING MEMORY

In this chapter, you will learn:

- About the types of physical memory housed on the system board and expansion boards
- How memory is used by DOS and Windows 9x
- How to manage memory using DOS and Windows 9x
- How to upgrade the memory in your computer

In the first three chapters, you saw memory chips and memory modules located on system boards and expansion cards and learned the basics of how software accesses this memory. In this chapter, you will learn how the operating system uses the memory on these boards, and how to manage this memory to meet the needs of applications software. You will also learn how to upgrade the RAM on your computer. Before turning to these topics, you will examine physical memory and its location, and what kinds of memory chips and modules are found in a computer.

Physical Memory

Recall that computer memory is divided into two categories: ROM and RAM. RAM is called primary memory, and temporarily holds data and instructions as the CPU processes them. All data stored in RAM is lost when the PC is turned off. (There is an exception; remember that one RAM chip, the CMOS setup chip, doesn't lose its data because it has its own battery, which powers it when the machine is turned off.) RAM is further divided into two categories: **static RAM (SRAM)** and **dynamic RAM (DRAM)**. ROM, on the other hand, stores system BIOS and startup BIOS programs in a microchip that does not lose data when the power is turned off. This section examines the different technologies of RAM and ROM, focusing on RAM technologies. Technology has steadily evolved to make RAM faster and increase its capacity (see Figure 4-1).

Besides the system board, recall that expansion boards can also have RAM chips to hold their data and ROM chips to provide the programming to drive their devices. For example, a network card contains ROM chips that provide the programming to communicate with the network. Similarly, video cards contain ROM chips with the programming that controls the monitor, and RAM chips to hold video data just before it is sent to the monitor.

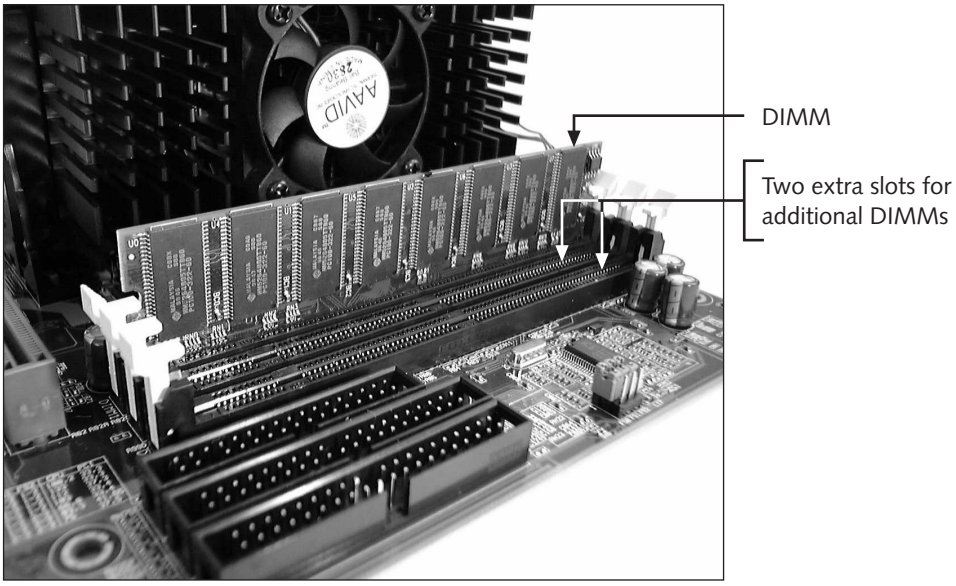


Figure 4-1 DRAM on most system boards today is stored on DIMMs

ROM on the System Board

Recall that ROM, or read-only memory, consists of memory chips that contain programs that are acid-etched into the chips at the factory (see Figure 4-2). The programs on a ROM chip (sometimes called firmware) are permanent; they cannot be changed. **EEPROM (electrically erasable programmable ROM)** chips (also known as Flash ROM chips),

do allow their programs to be changed. On EEPROM chips, a higher voltage is applied to a pin to erase its previous memory before a new instruction set or data is electronically written. **EPROM (erasable programmable ROM)** chips can also have their programs changed. They have a special window that allows the current memory contents to be erased with an ultraviolet light, so that the chip can be reprogrammed. However, for the discussions in this chapter, consider the EEPROM, EPROM, and ROM chips as providing BIOS that is not erasable during normal PC operations. In addition, when the text refers to ROM chips, they may be EEPROM, EPROM, or ROM chips.

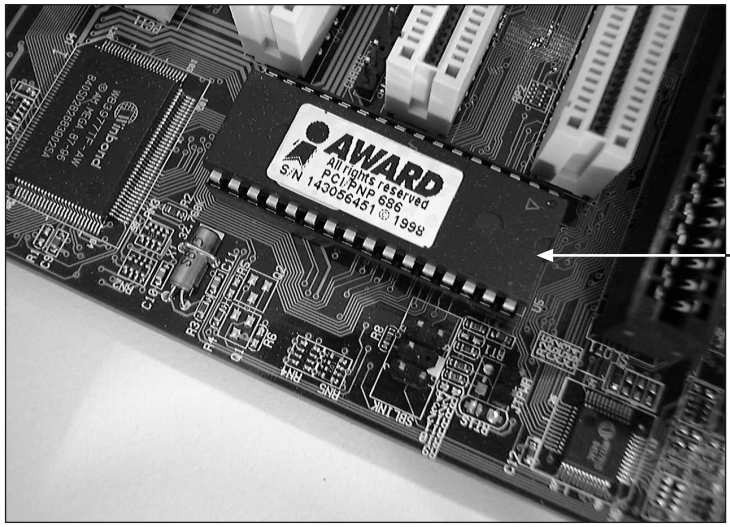


Figure 4-2 The ROM BIOS on newer system boards can be upgraded using software provided by the BIOS manufacturer

When you purchased your computer, it contained several ROM chips on the system board and some on the expansion boards. These ROM chips contain the programming that the computer uses to start up (boot itself) and to do routine utility operations, such as reading from and writing to hardware devices and performing basic data manipulation. As discussed in earlier chapters, the ROM chips on the system board contain much of the BIOS for your computer. System BIOS is a set of programs that do the basic input/output chores. The operating system calls on the System BIOS programs to interact with input/output devices as they are needed. Startup BIOS manages the early stages of the boot process.

ROM chips are usually socketed onto the system board. Occasionally, you must replace a ROM chip either because it is faulty or the ROM must be upgraded. You can easily remove the chips, and insert a new one into the socket.

Flash Memory

Recall that upgrading the programming on an existing chip is much easier than physically exchanging a ROM chip. Flash memory makes this possible. **Flash memory** acts more like secondary storage than like other types of memory because it does not lose its data when

the power is turned off. However, flash memory is different from a hard drive that holds its data as a magnetized area on a platter, because flash memory holds its data electronically. Also, flash memory is an electronic device and provides much faster data access than a mechanical device, such as a hard drive, though flash memory is more expensive.

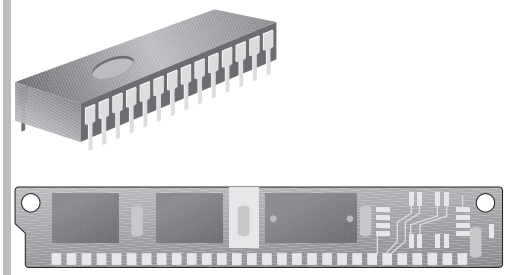
Flash memory uses EEPROM chips, and is used on notebook computers and PC Cards (PCMCIA), which look like thick credit cards and attach peripherals to notebook computers. Flash ROM is one example of this kind of memory on a system board. Flash memory is also used to hold picture data in digital cameras.

RAM on the System Board

A+CORE 1.1

Recall that besides ROM, the other kind of computer memory is RAM, random access memory. In the 1980s, RAM chips were either socketed or soldered directly on system boards, but today all RAM used as main memory is housed on SIMMs or DIMMs. Memory chips on SIMMs or DIMMs can only hold their data for a few milliseconds. Also recall that this memory, because it constantly needs refreshing, is called dynamic RAM (DRAM), pronounced "DEE-RAM."

Besides serving as main memory, RAM also provides a memory cache. A system board can have a lot of main memory to hold data and instructions as they are processed, and a little memory cache to help speed up the access time to main memory. Cache memory is contained on the system board or inside the CPU housing. On the system board, it is either on individual chips or on a memory module called a **COAST** (**cache on a stick**; see Figure 4-3). Recall that these memory chips hold their data for as long as the power is on and are therefore called static RAM or SRAM, pronounced "S-RAM." The recent trend has been to put all cache memory inside the CPU housing, but you still see some system boards that have cache memory.



A typical COAST module

Figure 4-3 SRAM on the system board is stored on single chips or COAST modules

DRAM and SRAM use several technologies, summarized in Table 4-1.

A+ CORE 1.1, 4.2

Table 4-1 Types of memory

Main Memory	Cache Memory
DRAM, needs constant refreshing	SRAM, does not need refreshing
Slower than SRAM because of refreshing time	Faster, but more expensive
Physically housed on DIMMs and SIMMs	Physically housed on the system board on COAST modules or single chips or included inside the processor case
Technologies include: • FPM • EDO • BEDO • Synchronous DRAM (SDRAM) • Direct Rambus DRAM • Double Data Rate SDRAM (SDRAM II) • SyncLink SDRAM (SLDRAM)	Technologies include: • Synchronous SRAM • Burst SRAM • Pipelined burst • Asynchronous SRAM • Housed within the processor case (new trend)
Memory addresses are assigned	No memory addresses assigned here

SRAM and Memory Caching

SRAM provides faster access than DRAM because data does not need to be constantly rewritten to SRAM, saving the CPU time used to refresh data in DRAM every 4 milliseconds or so. SRAM chips are made up of transistors that can hold a charge, but DRAM chips are made up of capacitors that must be recharged. But because SRAM chips are more expensive than DRAM chips, all RAM does not use SRAM chips. As a compromise, most computers have a little SRAM and a lot of DRAM.

Memory caching (see Figure 4-4) is a method used to store data or programs in SRAM for quick retrieval. Memory caching requires some SRAM chips and a cache controller. When memory caching is used, the cache controller anticipates what data or programming code the CPU will request next, and copies that data or programming code to the SRAM chips. Then, if the cache guessed correctly, it can satisfy the CPU request from SRAM without having to access the slower DRAM. Under normal conditions, memory caching guesses right more than 90% of the time and is an effective way of speeding up memory access.

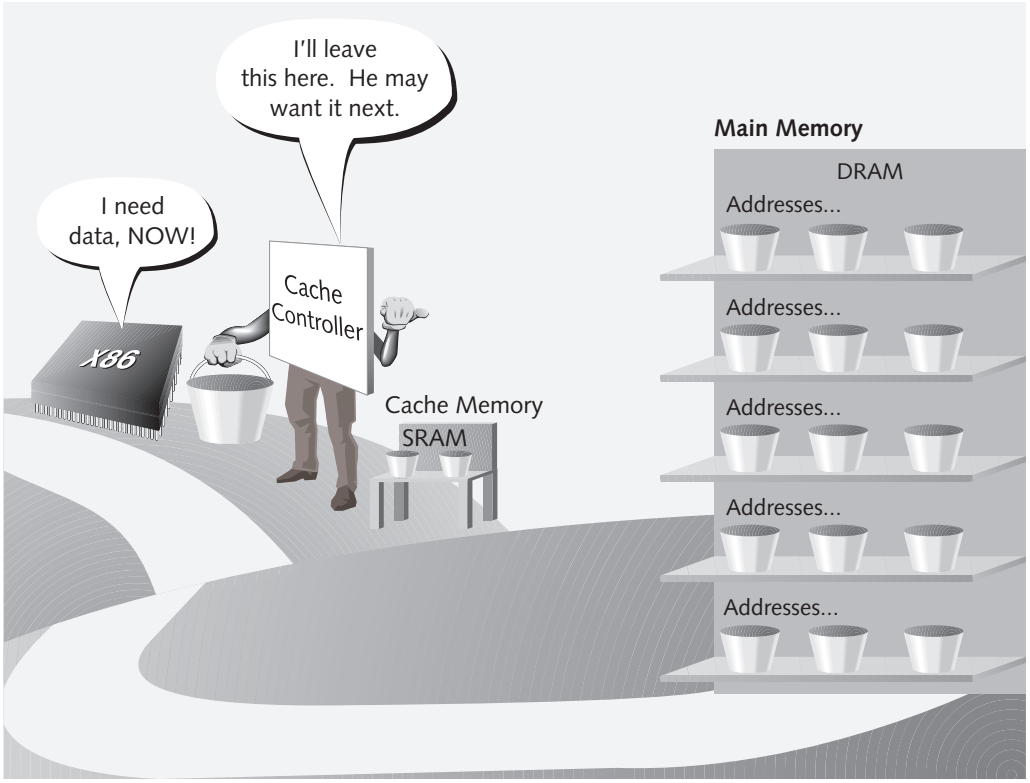


Figure 4-4 A memory cache (SRAM) temporarily holds data in expectation of what the CPU will request next

On older 386 computers, the cache controller was located on a single chip labeled the 385 chip, but 486 and later CPUs have the cache controller chip embedded in the CPU chip, housed together with some static RAM. This SRAM is called an internal cache, level 1 cache, or L1 cache, and is found in all 486 CPUs and higher. However, a system can have additional static RAM, called an external cache, level 2 cache, or L2 cache. This L2 cache may be on the system board of older computers or inside the CPU housing of newer systems.

If the cache memory is inside the CPU housing, you have no control over how much is present; it is dependent on the type of CPU you are using. However, older system boards were usually manufactured with some cache memory already installed, but with the option to add more in order to improve performance.

SRAM on the System Board

A+CORE 1.8

How much SRAM on a system board is enough without being prohibitively expensive? System boards often have 256K of SRAM installed with room for an additional 256K or more. The second 256K cache does not improve performance nearly as much as the first 256K does. For system boards running at 66 MHz, more than 512K of cache does not offer a significant improvement in access time. Figure 4-5 shows a system board that contains

256K of SRAM installed on the board in two single chips. A COAST slot is available to hold an additional 256K.

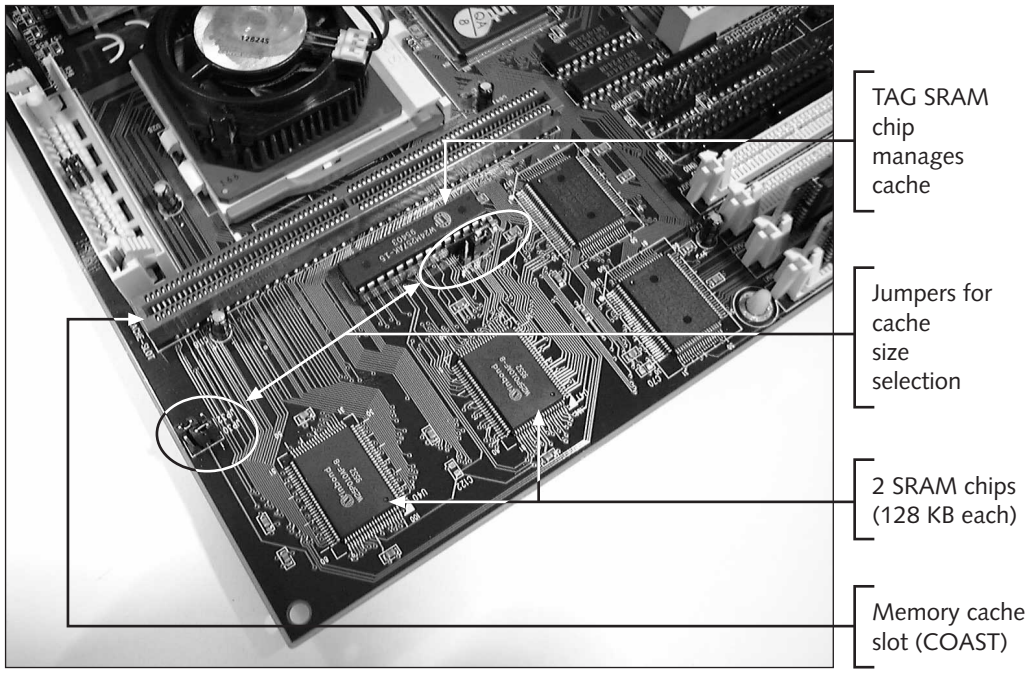


Figure 4-5 SRAM on this system board is stored in individual chips, and the board also has a COAST slot

A+ CORE 1.8 **Varieties of SRAM Memory on a System Board** SRAM is installed on a system board in increments of 64 KB, 128 KB, 256 KB, or 512 KB. See the documentation for the system board to determine which amounts of cache the board supports and what kind of memory to buy. Recall that all RAM sends data and control signals over the system bus. The methods used to coordinate how and when data and control signals are sent and read differ with various kinds of memory. SRAM uses a synchronous or asynchronous method (explained below). Synchronous SRAM is more expensive and about 30% faster than asynchronous SRAM.

A+ CORE 4.2 **Synchronous SRAM** requires a clock signal to manage or synchronize its control signals. The cache memory can then run in step with the CPU. Synchronous SRAM can be either burst or pipelined burst SRAM. Burst SRAM is more expensive than pipelined burst SRAM and only slightly faster. With **burst SRAM**, data is sent in a two-step process: first the data address, and then the data itself is sent. Burst SRAM sends a burst of data without sending all the addresses of the data, only the first address, allowing a large amount of data to be sent without interruption. **Pipelined burst SRAM** uses more clock cycles per transfer than does the burst without pipelining, but it does not significantly slow down the process and is desirable because of the reduced cost.

A+CORE 4.2

Asynchronous SRAM does not work in step with the CPU clock speed. It must look up the address sent to it by the CPU and return the data within one clock cycle, which makes it unable to process as much data in one request and results in overall slower memory access.

To understand the difference between asynchronous and synchronous memory, think of this analogy. Children are jumping rope with a long rope, and one child on each end turns the rope. A child who cannot keep in step with the turning rope can only run through on a single pass, and must come back around to make another pass. A child who can keep in step with the rope can run into the center and jump a while, until he or she is tired and runs out. Which child can perform the most rope-jumping cycles in a given amount of time? The one who can keep in step with the rope. Similarly, synchronous memory can retrieve data faster than asynchronous memory can because it keeps time with the system clock.

A system board can support both asynchronous and synchronous SRAM, but not at the same time. Before you buy new SRAM, check the documentation for the system board to determine which kinds of SRAM the board can support, and then see what kind of SRAM the board already has. You might need to replace the existing SRAM on the board to upgrade to a larger or faster cache.

Main Memory: SIMMs and DIMMs

In earlier PCs, main memory was stored on the system board as single, socketed chips, but today RAM is always stored in either SIMMs (single inline memory modules) or DIMMs (dual inline memory modules), which plug directly into the system board. The major difference between a SIMM and DIMM module is the width of the data path that the module accommodates. A SIMM has a data path of 32 bits, and a DIMM has a data path of 64 bits.

The technology used by SIMM and DIMM microchips has evolved to improve speed and size. SIMMs first used FPM and then EDO technologies, first discussed in Chapter 3. Next came DIMMs using burst EDO, followed by SDRAM technology and Direct Rambus technology. The goal with each new technology is to increase overall throughput. FPM is used on system boards that range in speed from 16 to 66 MHz. EDO is used on system boards rated at about 33 to 75 MHz, and SDRAM and Direct Rambus are used on system boards rated 66 MHz and higher. The older the system board, the older the memory technology it can use, so, as a PC technician, you must be familiar with all these technologies even though the boards sold today only use the latest.

SIMM Technologies Older SIMMs use FPM technology, have 30 pins on the edge connector, and come in sizes of 256K to 4 MB. Later FPM SIMMS had 72 pins on the edge connector and came in sizes up to 16 MB. Newer SIMMs use EDO technology, have 72 pins on the edge connector, are slightly longer, and come in sizes of 1 MB to 64 MB. A SIMM might have three or more chips, but this doesn't affect the amount of memory that the SIMM can hold. SIMMs are also rated by speed, measured in nanoseconds. A nanosecond (abbreviated ns) is one billionth of a second. Common SIMM speeds are 60, 70, or 80 ns. This speed is a measure of access time, the time it takes for the CPU to receive a value in response to a request. Access time includes the time it takes to refresh the chips. An access time of 60 ns is faster than an access time of 70 ns. Therefore, the smaller the speed rating, the faster the chip.

A+CORE 4.2

FPM (fast page mode) memory improved on the earlier memory types by sending the row address just once for many accesses to memory near that row. Earlier memory types required a complete row and column address for each memory access.

EDO (extended data output) memory is an improvement over earlier FPM memory. EDO memory is faster because it allows the memory controller to eliminate the 10-ns delay while it waited before it issued the next memory address. Without a memory cache, computer performance increases 10% to 20% when using EDO memory instead of FPM memory. However, if 256K of cache is used, the increased performance from FPM to EDO memory is only 1% to 2%. EDO memory does not cost significantly more than FPM memory, but your system board must be able to support it. Check your system board documentation or CMOS setup to determine if you should use EDO memory. If your system board does not support EDO memory, you can still use it, but it will not increase system performance. EDO memory is used on SIMMs and video memory, and is often used to provide on-board RAM on various expansion boards.

DIMM Technologies DIMMs are also rated by speed and the amount of memory they hold. DIMMs have 168 pins on the edge connector of the board and hold from 8 MB to 256 MB of RAM. The first DIMMs used EDO or burst EDO (BEDO) and then used **synchronous DRAM** (**SDRAM**) technology. Recall from Chapter 3 that **BEDO** is a refined version of EDO with improved access time over EDO. BEDO is not widely used today because Intel chose to not support it.

Either 3.3 volts or 5.0 volts can power SDRAM modules. Purchase DIMMs that use the voltage supported by your system board. Your system board also determines if you can use buffered, unbuffered, or registered DIMMs. Buffers are used by EDO DIMMs and registers are used by SDRAM DIMMs. Registers and buffers amplify a signal just before the data is written to the module. To determine which feature a DIMM has, check the position of the two notches on the DIMM module. In Figure 4-6, the position of the notch on the left identifies the module as registered (RFU), buffered, or unbuffered memory. The notch on the right identifies the voltage used by the module. The position of the notches not only helps identify the type of module, but also prevents the wrong kind of module from being used on a system board.

A+CORE 4.2

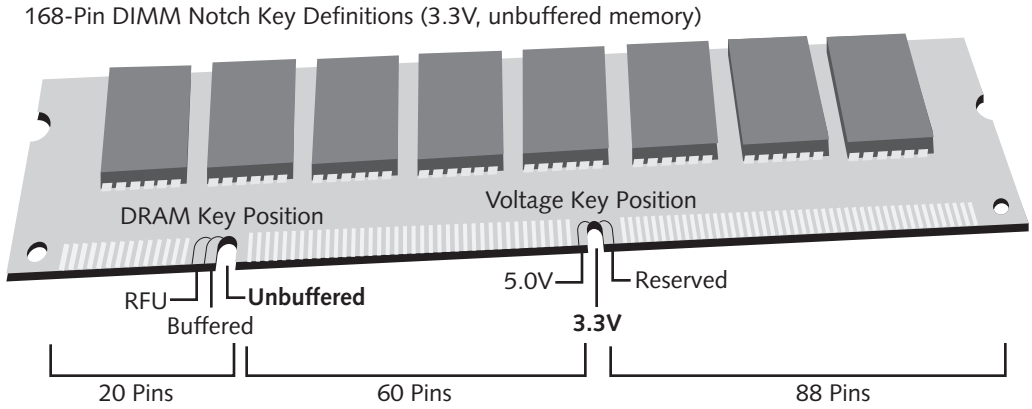


Figure 4-6 The positions of two notches on a DIMM identify the type of DIMM and the voltage requirement, and also prevent the wrong type from being installed on the system board

Other RAM Technologies Synchronous DRAM is currently the most popular memory type. SDRAM is rated by the system bus speed and operates in sync with the system clock, whereas older types of memory (FPM, EDO, and BEDO) all run at a constant speed. SDRAM currently comes in three variations: regular SDRAM, SDRAM II (DDR), and SyncLink (SLDRAM).

Regular SDRAM runs at the same speed as the system bus: 66 MHz, 100 MHz, 133 MHz, and so forth. The SDRAM data path is 64 bits wide, making SDRAM about 50% faster than its predecessor, EDO memory.

Double-data rate SDRAM (**DDR SDRAM**), sometimes called **SDRAM II**, runs twice as fast as regular SDRAM. Instead of processing data for each beat of the system clock as regular SDRAM does, it processes data when the beat rises and again when it falls, doubling the data rate of memory. If a system board is running at 100 MHz, then SDRAM II is running at 200 MHz with a data path of 64 bits. Future plans for DDR SDRAM include increasing this data path to 128 bits. DDR SDRAM is supported by a consortium of 20 major computer manufacturers. It is an open standard, meaning that no royalties need to be paid to use it. **SyncLink (SLDRAM)** was developed by a consortium of twelve DRAM manufactures. It improves on regular SDRAM by increasing the number of memory banks that can be accessed simultaneously from four to sixteen. Memory banks are discussed later in the chapter.

Direct Rambus DRAM (sometimes called **RDRAM** or **Direct RDRAM)** is named after Rambus, the company that developed it. The technology uses a narrow 16-bit data path rather than the wider 64-bit SDRAM data path. It works like a packeted network, not a traditional memory bus, and can run at speeds of 400 MHz to 800 MHz. The high speeds are possible because of the narrow bus width; wider buses cannot allow these high speeds. RDRAM uses a proprietary memory module called a RIMM and is not stored on a DIMM (see Figure 4-7). An earlier version of Rambus memory is Concurrent RDRAM, that is not as fast as Direct RDRAM. Manufacturers other than Rambus and Intel must pay licensing fees to use RDRAM, which might cause the industry to turn more toward SDRAM memory advancements over Rambus even though Intel is promoting RDRAM. The Intel 820

A+CORE 4.2, 4.4

and 840 chip sets both support RDRAM. Table 4-2 shows a comparison in data throughput of the current contenders for the memory market.

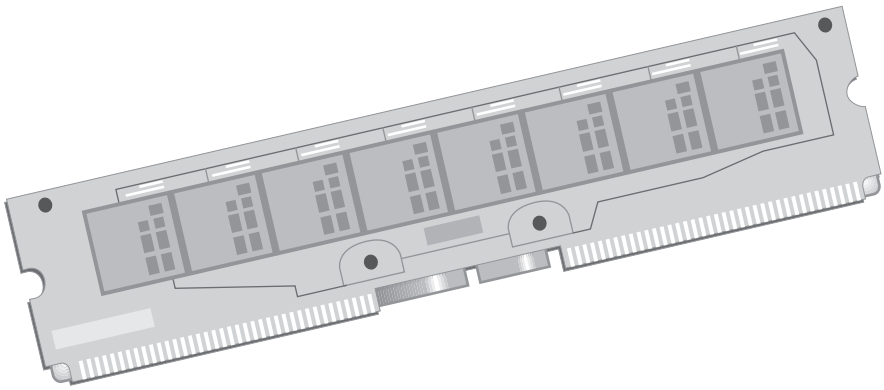


Figure 4-7 Direct Rambus DRAM is stored on a RIMM rather than a DIMM

Table 4-2 Comparing data throughput of current and future memory technologies

Memory Technology	Calculation of Throughput	Data Throughput
RDRAM	16 bits × 400 MHz	800 MB per second
RDRAM	16 bits × 800 MHz	1600 MB per second
SDRAM on 100 MHz board	64 bits × 100 MHz	400 MB per second
DDR-SDRAM on 133 MHz board	64 bits × 266 MHz	1064 MB per second
DDR-SDRAM on 166 MHz board	128 bits × 332 MHz	2656 MB per second

ECC, Parity, and Nonparity DRAM Some SDRAM memory modules support a chip set feature called **ECC (error checking and correction)**. DIMMs that support ECC have a ninth chip on the module (the ECC chip), whereas it normally has only eight chips. The module is identified as a 71 or 72-bit DIMM instead of a 64-bit DIMM. ECC uses an extra 7 or 8 bits to verify the integrity of every 64 bits stored on the module and correct any error, when possible. ECC memory costs more than regular memory, but is more reliable. To see if your system board supports ECC memory, look for the ability to enable or disable the feature in CMOS setup or check the system-board documentation.

Some older system boards support **parity memory**, and some only use **nonparity memory**. If a SIMM has an odd number of chips, most likely it is parity memory; an even number of chips usually indicates nonparity memory. As discussed in Chapter 3, parity memory validates the integrity of the data stored in RAM by counting the number of bits set to 1, which determines whether the total is an even or odd number. Parity memory then sets a parity bit either to make the number of 1-bits even (called even parity) or to make the number of 1s odd (called odd parity). When data is read, the number of 1s is counted. If parity is even but the number of 1s is odd (or vice versa), a parity error occurs, and the CPU stops processing the data.

A+CORE 4.2 Most manufacturers use nonparity memory to save processing time, and therefore money. Some of the issues surrounding memory parity are discussed next.

When a computer first boots up, the system must detect what type of memory is installed. To do so, the system can use two methods: Parallel Presence Detect (PPD) that uses resistors to communicate the type of memory present, or Serial Presence Detect (SPD) that stores information about the memory type in EPROM. When purchasing memory for a system, you must match the method used by the module to what the system board expects. See the system board documentation to know which type to buy. If the board does not specify the method used, assume PPD.

Another feature of memory is called **Cas Latency (CL)** and reflects the number of clock cycles that passes while data is written to memory. Values are 2 or 3 clock cycles. CL2 (Cas Latency 2) is a little faster than CL3 (Cas Latency 3). Again, use the memory type recommended by the system board manufacturer.

What to Look for When Buying Memory Chips and Modules

Memory chips and memory modules are sold at different speeds and sizes and use different technologies and features. Chips can be high-grade, low-grade, remanufactured, or used. Poor-quality memory chips can cause frequent **General Protection Fault (GPF) errors** in Windows, application errors, and errors that cause the system to hang—so it pays to know the quality and type of memory you are buying. The following are some guidelines to follow to ensure that you are purchasing high-quality memory chips.

Memory Speed Generally, use the fastest memory that your system board can support. The documentation for a system board states what speed of memory to use on the board, usually written as something like "Use 70 ns or faster." In this example, 60 ns will work on this board, but 80-ns memory will cause problems. It is possible, but not recommended, to mix the speed of memory modules on a system board, but don't mix the speeds within a single SIMM memory bank. You will learn more about this later in the chapter.

Tin or Gold Leads Memory modules and the banks that hold them can be either tin or gold. On a system board the connectors inside the memory slots are made of either tin or gold, as are the edge connectors on the memory modules. You should match tin leads to tin connectors and gold leads to gold connectors to prevent a chemical reaction between the two different metals, which can cause corrosion. Corrosion can create intermittent memory errors and even make the PC unable to boot.

Choosing the Correct Size of Module Not all sizes of memory modules fit on any one computer. Use the right number of SIMM or DIMM modules with the right amount of memory on each module to fit the memory banks on your system board. You can find more information about this later in the chapter.

Remanufactured and Used Modules Stamped on each chip of a SIMM or DIMM module is a chip ID that identifies the date the chip was manufactured. Look for the date in

the YYWW format, where YY is the year the chip was made and WW is the week of that year. For example, 9910 indicates a chip made in the 10th week of 1999. If you see date stamps on a SIMM or DIMM chip that are older than one year, these chips are probably used memory. If some chips are old, but some are new, the module is probably remanufactured. When buying memory modules, look for ones on which all chips have dates that are relatively close together and less than a year old.

Re-marked Chips New chips have a protective coating, which gives them a polished, reflective surface. If the surface of the chip is dull or matted, or you can scratch the markings off with a fingernail or knife, suspect that the chip has been re-marked. **Re-marked chips** have been used and returned to the factory, marked again, and then sold.

How DOS and Windows 9x View Memory

Now that you have reviewed the many ways memory can be physically installed on the system board and expansion boards, you can examine the logical organization of memory, a function of the OS. The following section explains how the OS categorizes, accesses, and uses memory.

Memory management under DOS and Windows 9x can seem complicated because of the way the process has evolved over the past 18 years or so. Like an old house that has been added to and remodeled several times, the present-day design is not as efficient as that of a brand new house. Decisions made by IBM and Microsoft in the early 1980s still significantly affect, and in some cases limit, the way memory is used today under Windows 98. Because Windows NT, followed by Windows 2000, has had the luxury of being designed from the ground up, they are free of those limitations.

Recall from Chapter 1 that earlier CPUs supported by DOS could only handle one program at a time, and that program had direct access to the first 1024K of memory addresses (see Figure 1-36 of Chapter 1). This processing mode is called real mode. Later, beginning with the i286 CPU, the CPU could manage more than one program at a time by switching among them (see Figure 1-37 of Chapter 1). Called protected mode, this mode allows for memory addresses above 1024K, 32-bit data transfers, and the OS managing access to memory for the programs. By managing memory, the OS could use the hard drive as virtual memory without a program's knowledge. However, needing to maintain backward compatibility with older real-mode applications software has slowed the transition from using real mode to protected mode. The process began with DOS, then continued to Windows 3.x and Windows 9x, and finally to Windows NT and Windows 2000, which provide a pure protected-mode OS whose program segments use 32-bit data flow.

Recall from Chapter 2 that earlier versions of DOS operated completely in real mode. Later, DOS added a memory manager extension (HIMEM.SYS) that allowed access to memory addresses above 1 MB. Early versions of Windows 3.x working with DOS could keep more than one program loaded at the same time by swapping them in and out of memory; this was called standard mode but was really real mode with some fancy maneuvering. Later versions of Windows 3.x let 16-bit programs share memory (more than one program could access the same memory and share its data) in a "virtual DOS machine" in 386 enhanced mode.

Windows 3.x could also perform virtual memory management by creating a swap file stored on the hard drive and using it as if it were memory.

Windows 95 was the first OS in this evolution of operating systems to support 32-bit protected mode applications software. Most of the OS code is written in 32-bit protected mode. It still allows 16-bit real-mode device drivers, and 16-bit software can run in a virtual DOS machine (VDM), just as with Windows 3.x, or can run in real mode. A VDM is an environment that a 32-bit protected-mode OS provides for a real-mode program to operate in.

Finally Windows NT made the break with the past. All its code is written in protected mode, and it does not allow other software to operate in real mode, but only in a virtual real mode that it tightly controls.

Table 4-3 summarizes the evolution of operating systems and software as it applies to memory. Notice that the first Microsoft OS to resolve the many issues involving real mode, particularly the 1024K limitation imposed by real mode, is Windows NT. With DOS and Windows 3.x, this limitation directly affects managing memory resources and, in some cases, can still be significant with Windows 9x. To explain how all this works, beginning with DOS and moving to Windows NT, you must understand basic memory concepts.

Table 4-3 Summary of how operating systems have evolved in managing memory

Operating System	Real Mode	Protected Mode
DOS	Operates totally in real mode, but later offered HIMEM.SYS, a device driver that allows programs access to extended memory	NA
DOS with Windows 2.x	Operated totally in real mode, but managed the process of switching programs in and out of memory	NA
DOS with Windows 3.x	Real mode is called standard mode. Allows only one 16-bit application at a time in memory	Protected mode is called 386 enhanced mode. Multiple applications can share memory. 16-bit applications share a virtual machine.

A+ OS 1.1

Table 4-3 Summary of how operating systems have evolved in managing memory (continued)

Operating System	Real Mode	Protected Mode
Windows 9x	Allows real-mode drivers to be loaded during startup. 16-bit DOS applications are allowed a real-mode session.	Switches back and forth between real mode and protected mode as necessary. Supports both 16-bit and 32-bit applications in a virtual machine.
Windows NT and Windows 2000	NA	All work is done in protected mode. Supports 32-bit applications. 16-bit applications can operate in a virtual machine only.

Physical Memory and Memory Addresses

The terms "memory" and "memory address" have significantly different definitions. **Memory** is physical microchips that can hold data and programming. It is located on the system board or on circuit boards as single chips or modules. Memory can be either ROM or RAM. A **memory address**, on the other hand, is a number the CPU assigns to ROM or RAM to track the memory it can use. Recall that a CPU has a limited number of memory addresses that it can assign to physical memory, determined by the number of memory address lines available on the memory bus.

Recall also that some older 16-bit programs work only when they can use certain memory addresses, such as a hexadecimal number like C80000. This address is part of the physical memory's ROM programming; no other address works for it. An example of this kind of memory is a ROM chip on an older video card. Some memory, usually ROM chips on expansion boards, must be assigned one of two—or sometimes three—sets of addresses. You make the choice by setting jumpers or DIP switches on the board or, in more recent cases, when you run an installation program for the board. New Plug and Play boards don't have this restriction. Their ROM code can be assigned any values chosen by the OS or system BIOS. Plug and Play cards are required to use whatever memory addresses are assigned to them. The system is free to assign any address it chooses to this physical memory.

Both RAM and ROM must be assigned memory addresses so the CPU can access this memory. System BIOS stored on system-board ROM chips must be assigned addresses by the CPU so that the CPU can access that programming. The assigning of addresses to both RAM and ROM occurs during booting, and is sometimes called **memory mapping**. In Figure 4-8, the memory addresses available to the CPU are listed on the left, and the physical RAM and ROM needing these addresses are on the right. These RAM and ROM chips and modules are located on the system board, a video card, and a network card in this example.

Programming stored on ROM chips is not usually copied into RAM, despite what many people believe. It is simply assigned memory addresses by the CPU. These ROM programs become part of the total memory available to the CPU, and do not use up part of total RAM. The resources they use are memory addresses. The RAM memory is still available to be assigned other addresses. **Shadowing ROM**, discussed in Chapter 3, is an exception because the programs stored on ROM are copied to RAM when the ROM is shadowed to improve performance.

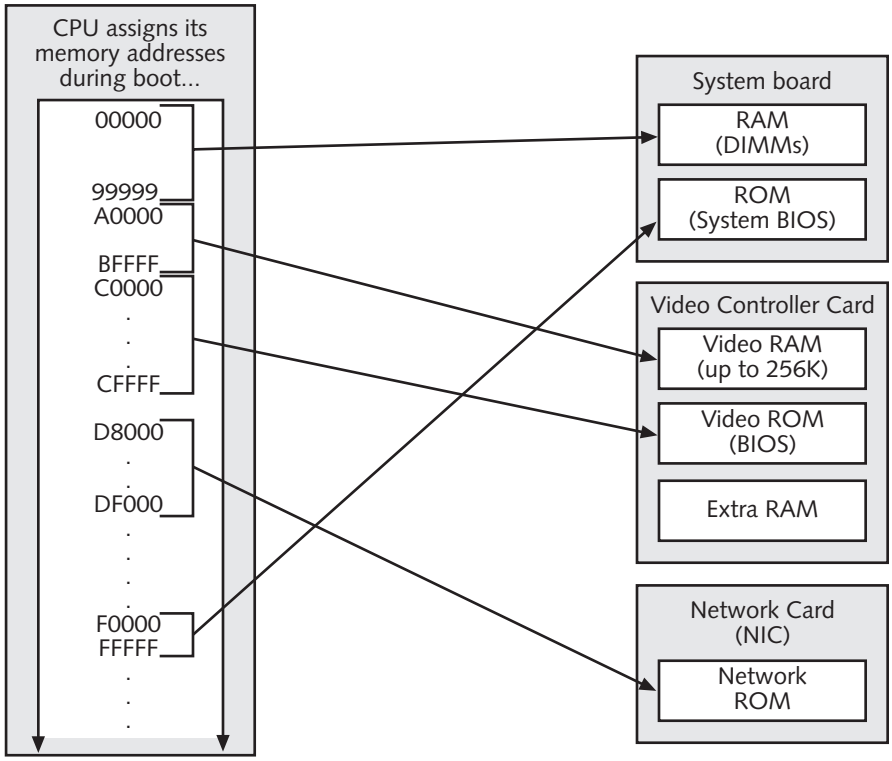


Figure 4-8 CPU assigns memory addresses during booting

Finally, memory management in DOS and Windows 9x presents limitations not so much because of the operating systems themselves, but because their applications used the standards presented to the industry when DOS was first introduced. DOS (and the applications written for it in the 1980s that are still used today) incorporated assumptions that continue to affect memory management.

Compared to that of other operating systems, memory management in DOS and Windows 9x is handicapped because DOS has existed longer than most other operating systems. Therefore, DOS must maintain compatibility with software that has been around for a long time. Also, Microsoft made the commitment with Windows 9x that it, too, would be compatible with

older software written for DOS and Windows 3.x using DOS. Probably the greatest limitation of DOS today is this commitment to maintain backward compatibility with older software and hardware.

Areas of the Memory Map

A+ OS 1.1

The following section introduces several types of memory that the OS manages: conventional, upper, extended, and expanded memory. Except for expanded, these memory types are logical divisions or categories rather than physical ones, and the divisions are determined by their memory addresses rather than by their physical location. The following section covers this logical **memory management**. A segment of RAM can be assigned memory addresses in the upper memory range today but be assigned a range of addresses in extended memory tomorrow. It's still just RAM, no matter what address it's assigned. The difference is the way the CPU can use this memory because of the addresses assigned to it.

To get a clear picture of this memory-addressing schema, consider the memory map shown in Figure 4-9. The first 640K of memory addresses are called **conventional memory**, or **base memory**. The memory addresses from 640K up to 1024K are called **upper memory**. Memory above 1024K is called **extended memory**. The first 64K of extended memory is called the **high memory area (HMA)**. Memory that is accessed in 16K segments, or pages, using a window in upper memory is called **expanded memory**.

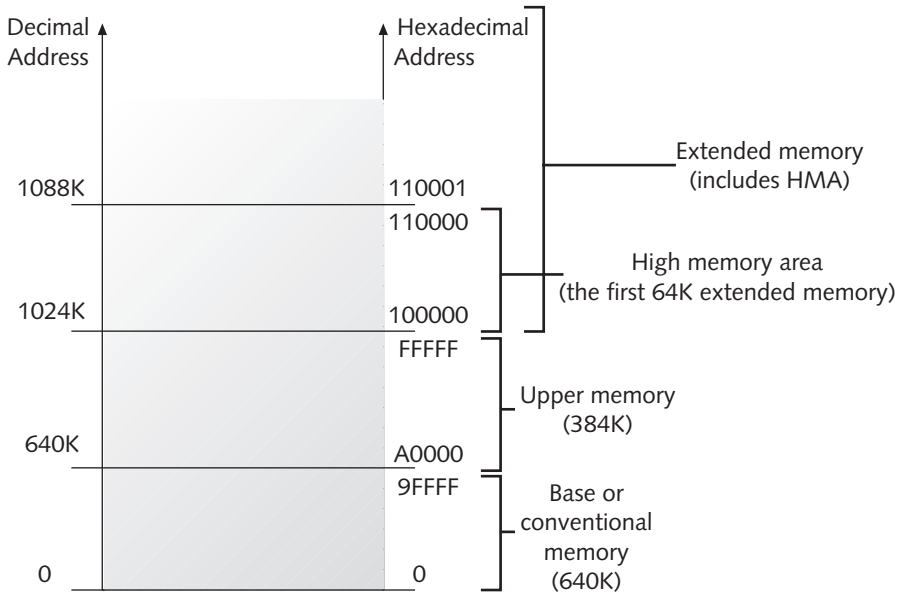


Figure 4-9 Memory address map showing the starting and ending addresses of conventional, upper, and extended memories, including the high memory area

A+ OS 1.1

Conventional Memory

In the early 1980s, when IBM and Microsoft were designing the original PCs, they decided to make 640K of memory addresses available to the user, thinking that this was plenty for anything the user would ever want to do. This 640K of addresses were intended to hold the OS, the applications software, and the data being processed. At that time, 640K of memory addresses were many times more than enough to handle all the applications available. Today, 640K of memory addresses are inadequate, for the following reasons:

- Many applications are very large programs, requiring a considerable number of memory addresses to hold the programs as well as the data.
- Often more than one application is run at a time, each of which requires its own memory area for the program and data. Also, sometimes computers in a network serve more than one user at a time. In the early 1980s, a PC was expected to be used by a single user, operating one application at a time.
- Users expect software to provide a friendly graphical user interface, or GUI. Graphical user interfaces provide icons, graphics, and windows on a screen, all requiring large amounts of memory.

The problem caused by restricting the number of memory addresses available to the user to only 640K could have been easily solved by simply providing more addresses to the user in future versions of DOS. However, another original design decision ruled this out. The next group of memory addresses, the 384K above conventional memory, called upper memory, were assigned to utility operations for the system. The system requires memory addresses to communicate with peripherals. The programs (such as BIOS on the expansion boards) and data are assigned memory addresses in this upper memory area. For example, the video BIOS and its data are placed in the very first part of upper memory, the area from 640K to 768K. All video ROM written for DOS-based computers assumes that these programs and data are stored in this area. Also, many DOS and Windows applications interact directly with video ROM and RAM in this address range.

Programs almost always expect data to be written into memory directly above the addresses for the program itself, an important fact for understanding memory management. Thus, if a program begins storing its data above its location in conventional memory, eventually it will "hit the ceiling," the beginning of upper memory assigned to video ROM. The major reason that applications have 640K memory limit is that video ROM begins at 640K. If DOS and Windows 9x allowed applications into these upper memory addresses, all DOS-compatible video ROM would need to be rewritten, many existing video boards would be obsolete, and many DOS applications that access these video addresses would not work.

Windows NT and Windows 2000 are not backward compatible with older hardware and software because they do not manage memory the same way. You will learn more about how these operating systems manage memory later in this book.

Sixteen-bit applications under DOS and Windows 9x are boxed in the first 640K of addresses. Also, because the OS, BIOS, and TSRs use some conventional memory, not all the 640K of memory is available to applications. Several methods have been proposed and used

to expand the 640K limit. These methods involve either freeing more conventional memory or providing memory outside of conventional memory. The ultimate objective is to provide more memory addresses to application programs and their data.

Upper Memory

A+ OS 1.1

The memory map in Figure 4-9 shows that the memory addresses from 640K up to (but not including) 1024K are called upper memory. In the hexadecimal number system (see Appendix D for an explanation of this system), upper memory begins at A0000 and goes through FFFFF. Video ROM and RAM are stored in the first part of upper memory, hex A0000 through CFFFF (the A, B, and C areas of memory). BIOS programs for other expansion boards are assigned memory addresses in the remaining portions of upper memory. BIOS on the system board (system BIOS) is assigned the top part of upper memory from F0000 through FFFFF (the F area of upper memory). Upper memory often has unassigned addresses, depending on which boards are present in the system. Managing memory effectively involves gaining access to these unused addresses in upper memory and using them to store device drivers and TSRs.

Figure 4-10 shows that video memory addresses fall between A0000 and CFFFF. For VGA and Super VGA video, the A and B areas hold data sent to the video card, and the C area contains the video BIOS.

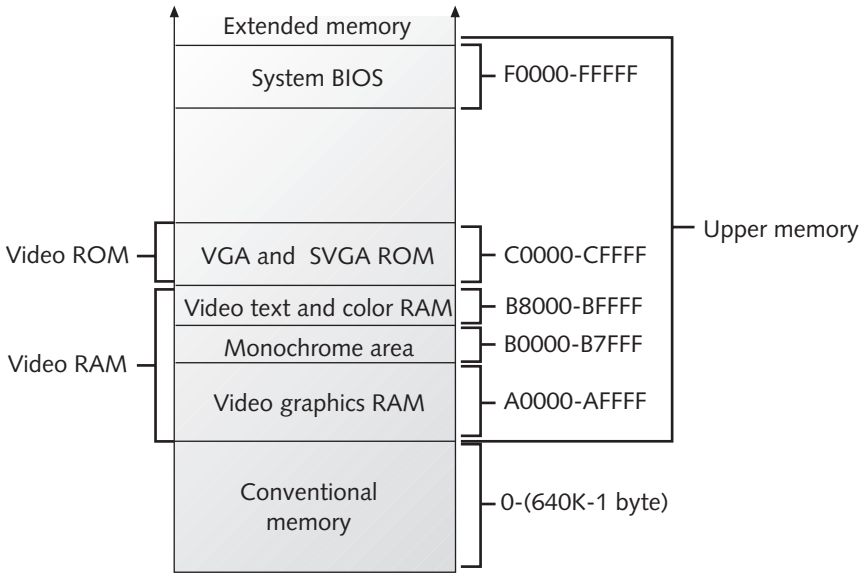


Figure 4-10 Memory map of upper memory showing starting and finishing addresses and video ROM and RAM assignments

A+ OS 1.1

Extended Memory and the High Memory Area

Memory above 1 MB is called extended memory. The first 64K of extended memory is called the high memory area, and exists because a bug in the programming for the older 286 CPU (the first CPU to use extended memory) produced this small pocket of unused memory addresses. Beginning with DOS 5, the OS capitalized on this bug by storing portions of itself in the high memory area, thus freeing some conventional memory where DOS had been stored. This method of storing part of DOS in the high memory area is called "loading DOS high." You will see how to do this later in the chapter.

Extended memory is actually managed by the OS as a device (the device is memory) that is controlled by a device driver. To access extended memory, you need the device driver (called a memory manager) that controls it, and you must use applications that have been written to use the extended memory. DOS 5+ and Windows 3+ both offer memory management software that you will learn to use later in the chapter, and Windows 9x has automated the process. The amount of extended memory you can have on your computer is limited by the amount of RAM that can be installed on your system board and the number of memory addresses the CPU and the memory bus can support.

Expanded Memory

Few of today's applications software programs use expanded memory, but it is included in our discussion because occasionally you see an older program that needs it, and mainly because it is similar to the Windows NT memory model. Figure 4-11 shows that expanded memory falls outside of the linear addressing of memory. Linear addressing means that all memory is assigned addresses beginning with 1, and counting up. Expanded memory breaks out of this pattern. When first introduced in the 1980s, expanded memory was always physically located on the memory expansion boards that also contained the ROM BIOS that managed it. Expanded memory was made available through a small window called a page frame. A **page frame** is 64K of upper memory projected onto expanded memory in 16K segments called pages. It works as follows.

The system requests some expanded memory from the memory manager. The memory manager selects some expanded memory on the expanded memory card, say 32K of it, and assigns 32K of the 64K of upper memory addresses to this memory. The system now has actual physical memory (on the expanded memory card) assigned to the memory addresses in upper memory. The expanded memory manager keeps track of what memory on the card is being used, so that, on the next request for expanded memory, it will assign upper memory addresses to a different 32K of physical memory on the card. The same memory addresses in upper memory can be used over and over to access different memory addresses on the expansion board. These 64K of memory addresses can access several hundred kilobytes of physical memory during a single session. The relatively small 64K of upper memory addresses becomes the window that is moved around, over the larger amount of expanded memory, thus gaining access to it. (Think of this window as a picture frame moving over a large area, showing only a small space at any one time.) Applications software must be written in a way that makes use of this expanded memory.

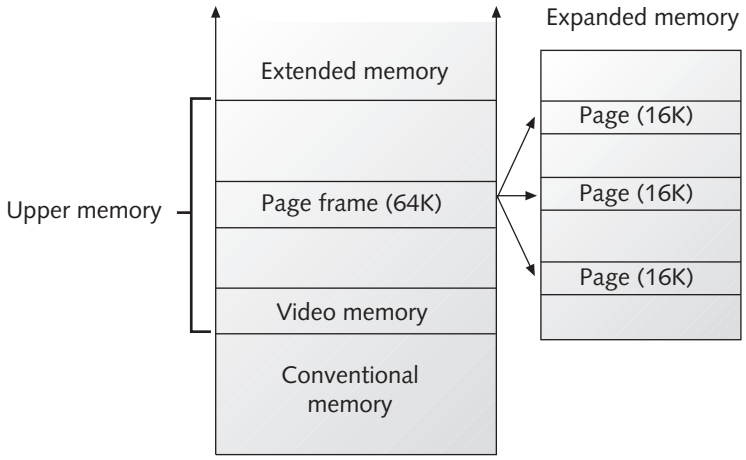


Figure 4-11 Expanded memory map showing page frame and pages; the page frame serves as a "window" into expanded memory

A+ OS 1.1

Expanded memory was developed by the Lotus Corporation in conjunction with Intel and Microsoft, because version 2.0 of Lotus 1-2-3 needed more memory. Other applications also make use of expanded memory. The term LIM (Lotus, Intel, Microsoft) memory is sometimes used for expanded memory. Incidentally, later versions of Lotus 1-2-3 don't use expanded memory, but instead use extended memory. Expanded memory is also sometimes called EMS memory (expanded memory specifications).

At one time, to have expanded memory for an application that required it, you installed an expanded memory card on your computer. There is, however, an alternative to an expanded memory card. The Windows and DOS extended memory manager for the i386 and later CPUs can emulate expanded memory by taking some extended memory and having the application treat this extended memory as expanded memory. You can then make either extended memory or emulated expanded memory available to your software, depending on which kind of memory the software requires.

Virtual Memory

A+ OS 1.1

Virtual memory is using hard drive space so that it acts like memory. Windows stores virtual memory in a file called a swap file, discussed later in the chapter. The purpose of virtual memory is to increase the amount of memory available. Of course, virtual memory works at a considerably slower speed than real memory, and uses hard drive space. For example, a hard drive may have a data access time of 10 ms (10/1,000,000 second), whereas RAM speed may be 60 ns (60/1,000,000,000 second).

For an OS to use virtual memory, it must be operating in protected mode. DOS and earlier versions of Windows work only in real mode. Windows 3.x uses 386 enhanced mode, which is protected mode and, therefore, provides virtual memory to applications.

RAM Drives

Creating a **RAM drive** makes part of memory act like a hard drive. A RAM drive is just the opposite of virtual memory. By loading the TSR that creates a RAM drive, you can make your computer appear to have a second hard drive, such as drive D. This drive D is really an area of memory set aside to act like a drive. A RAM drive was once useful to hold software that had to be loaded many times. Since software is so large today, RAM drives have become impractical. However, RAM drives still have value in a few instances, as you will see later. Since a RAM drive is not a real drive, but memory, nothing is really saved when data is stored to a RAM drive. Data written to a RAM drive is no safer than data still in RAM. Therefore, do not use a RAM drive to "store" data.

Another purpose for a RAM drive is to store programs. Programs are permanently stored on a real hard drive, such as drive C. On startup, the programs are copied to the RAM drive, such as drive D. The program is loaded from drive D (one part of memory) to another part of memory, where it is executed. (Remember that programs can't be executed from a hard drive, even if the hard drive is not a drive at all, but a RAM drive.) Loading the program from the RAM drive is faster than loading the program from the hard drive, because it is faster to copy from memory to memory than from the hard drive to memory.

The only gain in speed in this example is in the loading process. Once the software is loaded, the RAM drive is no longer used. You gain speed from using a RAM drive if you load the same software many times in one session.

With DOS and Windows 3.x, you create a RAM drive by using a DEVICE= command in the CONFIG.SYS file:

```
DEVICE=C:\DOS\RAMDRIVE.SYS 1024
```

The program that produces the RAM drive, RAMDRIVE.SYS, is stored on drive C in the \DOS directory. The 1024 in the command line tells DOS to create a RAM drive that contains 1024K of space. The RAMDRIVE program assigns the next available letter to the drive. Suppose you have a floppy disk drive and a hard drive, which are drives A and C respectively. The RAM drive then becomes drive D and is used just like any other drive in the system.

Summary of How Memory is Managed

This section introduced the different kinds of memory and the ways it can be used. Memory management makes the greatest amount of conventional memory available to an application. During the boot process, after ROM and RAM from expansion boards acquire upper memory addresses, any unused addresses in upper memory are used to hold TSRs and device drivers. Also, applications must be able to access extended and expanded memory. The next section details how DOS and Windows 9x manage memory effectively.

Managing Memory with DOS

A+ OS 1.1

Beginning with DOS 5.0, memory management under DOS included access to upper and extended memory for device drivers and applications software. The TSRs that manage memory above 640K, HIMEM.SYS and EMM386.EXE, are loaded and used from the CONFIG.SYS and AUTOEXEC.BAT files at startup. The program file **HIMEM.SYS** is the device driver for all memory above 640K. The program file **EMM386.EXE** contains the software that loads user TSRs and device drivers into upper memory and makes extended memory emulate expanded memory for those applications that use expanded memory.

Using HIMEM.SYS

HIMEM.SYS is loaded into your system from the CONFIG.SYS file. Recall that the CONFIG.SYS file is executed during booting and loads device drivers into memory. Next, COMMAND.COM is executed and, in turn, runs AUTOEXEC.BAT. AUTOEXEC.BAT is the DOS batch file that automatically runs each time you boot the computer. It contains a list of DOS commands that are executed by COMMAND.COM. TSRs can be launched from AUTOEXEC.BAT. Remember that a TSR is a terminate-and-stay-resident program that is loaded into memory and lies dormant until activated by some later action, such as pressing a "hot" key.

HIMEM.SYS is considered a device driver because it manages a device, such as memory. It is executed by the DEVICE= command in CONFIG.SYS. Figure 4-12 shows an example of a very simple CONFIG.SYS file that loads HIMEM.SYS.

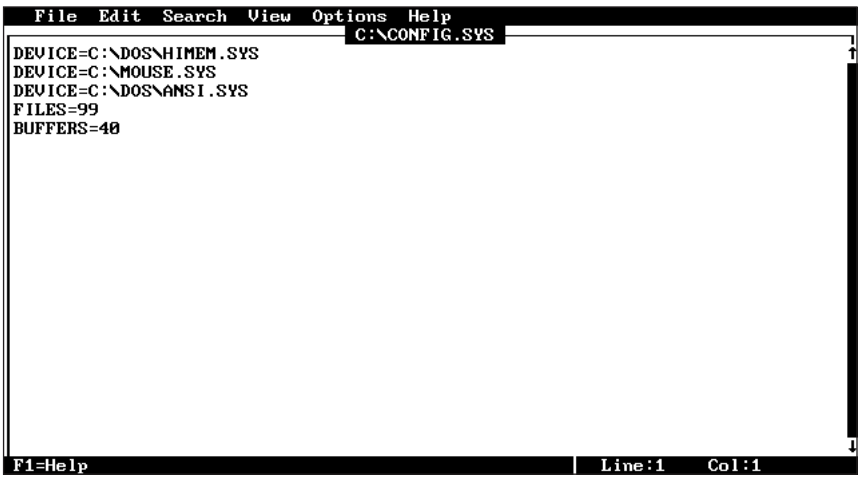


Figure 4-12 CONFIG.SYS set to use memory above 640K

A+ OS 3.2

In Figure 4-12, you can see the DOS EDIT command, which provides a full-screen text editor that can create or edit a file. To access CONFIG.SYS, at a DOS prompt or a Run dialog box in Windows 9x, enter this command:

```
EDIT C:\CONFIG.SYS
```

To exit from the editor, press the **ALT** key to activate the menus, and choose **EXIT** from the File menu. When asked if you want to save your changes, respond **YES** to exit the editor and save any changes. After editing the CONFIG.SYS file, boot the computer to activate the changes.

The last two lines in Figure 4-12 are usually found in a CONFIG.SYS file. The command FILES=99 tells DOS how many files it can have open at one time. The command BUFFERS=40 tells DOS how many buffers to maintain to transfer data to and from secondary storage devices. A **buffer** is an area of memory that serves as a holding area for incoming or outgoing data. When writing data to a device, DOS fills the holding area with data. When the area is full, DOS writes all the data at once to the device. When reading data, DOS reads enough data from the device to fill the holding area, or buffer, and uses all that data before going back for more. Buffers reduce the need for DOS to return again and again to the device for data; DOS only needs to access the device occasionally for data. Using buffers to speed up reading and writing data is largely outdated today. Disk caching, which is described in later chapters, is more common and effective. However, many software packages still use buffers, and so DOS must provide these buffers. The more files and buffers that are open, the less free memory is available, so limit the number of buffers to the smallest value that will still accommodate the DOS applications.

A+ OS 1.1

The first line in the CONFIG.SYS file is the one that loads the driver HIMEM.SYS into memory. HIMEM.SYS makes memory above 640K available to your applications as extended memory. Remember that you must be using applications software that can use extended memory.

The first line in the CONFIG.SYS file also contains the path that tells DOS where to find HIMEM.SYS. The path C:\DOS\ indicates that HIMEM.SYS should be stored on drive C in the directory named \DOS. If you are using the Windows version of HIMEM.SYS, your command line looks like this:

```
DEVICE=C:\WINDOWS\HIMEM.SYS
```

You should use the latest version of HIMEM.SYS that you have. Look in the \DOS or \WINDOWS directory, and use the latest file.

The second line in the CONFIG.SYS file, DEVICE=C:\MOUSE.SYS, tells DOS to load a device driver into memory. The driver that controls a mouse is stored in the file named MOUSE.SYS on drive C in the root directory. The mouse software is a TSR that is loaded into conventional memory; it does not execute until you use the mouse. The third line in the CONFIG.SYS file, DEVICE=C:\DOS\ANSI.SYS, tells DOS to load the device driver ANSI.SYS into memory. ANSI.SYS helps control the keyboard and monitor, providing color on the monitor and an additional set of characters to the ASCII character set. For more information about ASCII and ANSI, see Appendix C.

Using EMM386.EXE

A+ OS 1.1

In DOS and Windows, EMM386.EXE manages the memory addresses in upper memory and also emulates expanded memory. To provide the maximum amount of conventional memory to applications, store as many programs, TSRs, and device drivers as possible in upper memory, using upper memory blocks. To see which programs are loaded into conventional memory and which programs are loaded into upper memory, use a variation of the MEM command with the /C option. Also include the |MORE option to page the results to your screen. Figure 4-13 was produced using this command:

```
MEM /C |MORE
```

In Figure 4-13, the first column shows the programs currently loaded in memory. The second column shows the total amount of memory used by each program. The columns labeled Conventional and Upper Memory show the amount of memory being used by each program in each of these categories. This PC is not making use of upper memory for any of its programs. At the bottom of the screen is the total amount of free conventional memory (578,448 bytes) that is available to new programs to be loaded. Making this value as high as possible is the subject of this section.

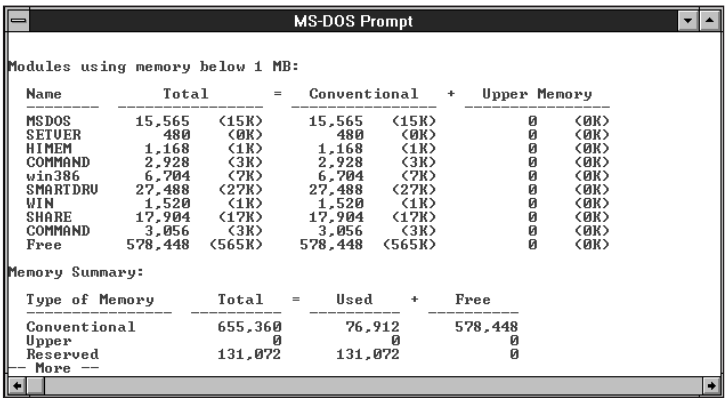


Figure 4-13 MEM report with /C option on a PC not using upper memory

Creating and Using Upper Memory Blocks

Figure 4-14 shows an example of a CONFIG.SYS file that is set to use upper memory addresses. The first line loads the HIMEM.SYS driver. The second line loads the EMM386.EXE file on drive C in the \DOS directory. EMM386.EXE assigns addresses in upper memory to memory made available by the HIMEM.SYS driver. The NOEMS switch at the end of the command line says to DOS, "Do not create any simulated expanded memory." The command to load EMM386.EXE must appear after the command to load HIMEM.SYS in the CONFIG.SYS file.

A+ OS 1.1

The command DOS=HIGH,UMB serves two purposes. The one command line can be broken into two commands like this:

```
DOS=HIGH
DOS=UMB
```

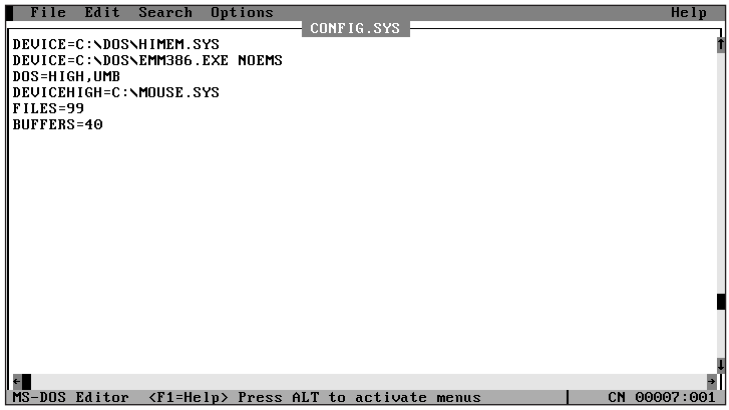


Figure 4-14 CONFIG.SYS set to use upper memory

The DOS=HIGH portion tells DOS to load part of DOS into the high memory area. Remember that the high memory area is the first 64K of extended memory. This memory is usually unused unless we choose to store part of DOS in it with this command line. Including this command in CONFIG.SYS frees some conventional memory that would have been used by DOS. Storing DOS in this high memory area is sometimes called "loading DOS high."

The second part of the command, DOS=UMB, creates upper memory blocks. An **upper memory block (UMB)** is a group of consecutive memory addresses in the upper memory area that has had physical memory assigned to it. DOS identifies blocks that are currently not being used by system ROM or expansion boards, and the memory manager makes these blocks available for use. This command, DOS=UMB, enables DOS to access these upper memory blocks. After the UMBs are created, they can be used in these ways:

- DEVICEHIGH= command in CONFIG.SYS
- LOADHIGH command in AUTOEXEC.BAT
- LOADHIGH command at the DOS prompt (explained below)

The next line in the CONFIG.SYS file in Figure 4-14 uses a UMB. The command DEVICEHIGH=C:\MOUSE.SYS tells DOS to load the mouse device driver into one of the upper memory blocks created and made available by the previous three lines. This process of loading a program into upper memory addresses is called **loading high**.

Loading Device Drivers High

A+ OS 3.2

Using the DEVICEHIGH= command in CONFIG.SYS, rather than the DEVICE= command, causes the driver to load high. With the DEVICEHIGH command, DOS stores these

A+ OS 3.2

drivers in UMBs using the largest UMB first, then the next largest, and so on until all are loaded. Therefore, to make sure there is enough room to hold them all in upper memory, order the DEVICEHIGH= command lines in CONFIG.SYS so that the largest drivers are loaded first. To determine the size of a driver, begin by simply looking at the size of the file. However, other factors also affect the size of a UMB needed for a driver.

A device driver needs some space immediately above it to hold its data. In addition, sometimes when the program first executes, it requires some extra room to initialize itself. This space may be released later, but it is still required in order to execute the driver. The largest amount of memory that a driver needs to initialize itself and to hold its data is called the maximum size, or the **load size**. It is almost always a little larger than the size of the program file. Since you don't always accurately predict what this maximum size will be, you can understand why a device driver might fail to load into upper memory even when the UMB is larger than the file size. If you include the DEVICEHIGH= command in CONFIG.SYS, but later discover that the device driver is still in conventional memory and did not load high, DOS did not find a UMB large enough for it.

You can determine the amount of memory a device driver allocates for itself and its data by using the DOS MEM command with the /M filename option:

```
MEM /M filename
```

The filename is the name of the device driver without the file extension. Later, this chapter will examine methods to determine just where in upper memory a driver is loaded, and what to do if things go wrong.

Loading TSRs High

Recall that a TSR is any program that remains loaded in memory, waiting to be executed at a later time. All device drivers are TSRs, but not all TSRs are device drivers. You can load TSRs that are not device drivers into upper memory either from AUTOEXEC.BAT or from the DOS prompt. In either case, the command is LOADHIGH followed by the path and name of the program file.

For example, suppose you want to print some screen captures like the screen capture that was used to produce Figure 4-13. While using DOS, you can press the Print Screen key to get a printout of the screen; while using Windows 3.x, use the Print Screen key to copy the contents of a screen into the clipboard, and print it using an applications software program such as Paint Brush. Sometimes the printout, however, looks very rough, is hard to read, and cannot easily be saved to a file. The screen capture in Figure 4-13 was created and printed with a software package called Pizazz Plus. Pizazz Plus is a TSR. It is loaded into memory by typing C:\PZP\PZP at the C prompt. This command tells DOS the name of the program file, PZP, and the path to it, drive C, \PZP directory. The program loads into conventional memory and immediately returns to the C prompt. It does not execute at the time it is loaded; it remains dormant until you press the Print Screen key.

You can then execute any software package and work until you are ready to use the screen capture software. When you have a screen ready to print, press the **Print Screen** key, the hot key for this TSR. Rather than printing the screen, the Print Screen key brings up the Pizazz menu,

and you can save the screen to a file for printing later. When you exit from the Pizazz menu, DOS returns you to the software you are executing, and you pick up where you left off.

The Pizazz program, like all DOS applications, expects to use conventional memory, and a problem arises when you are running other software that requires a lot of conventional memory. The software might not run at all, might run unacceptably slowly, or might give errors. You can solve this problem by storing the Pizazz program in upper memory instead of in conventional memory. You do this by using the following command at the C prompt:

```
C:\> LOADHIGH C:\PZP\PZP
```

The command line can be shortened by replacing LOADHIGH with the shorter version, LH:

```
C:\>LH C:\PZP\PZP
```

The program is loaded into the largest UMB available and does not use up more precious conventional memory. Note that before the LOADHIGH command will work, these three lines must be added to CONFIG.SYS and executed by booting the computer:

```
DEVICE=C:\DOS\HIMEM.SYS
DEVICE=C:\DOS\EMM386.EXE NOEMS
DOS=UMB
```

If you intend to use the Pizazz software often, you can make it load into upper memory every time you boot the computer. To do this, the previous LH line is included in the AUTOEXEC.BAT file without any changes.

Figure 4-15 shows an AUTOEXEC.BAT file that loads the Pizazz software high every time the computer boots (LH C:\PZP\PZP). Note that the path to the PZP program must be included in the last command line, because \PZP is not included in the earlier PATH command.

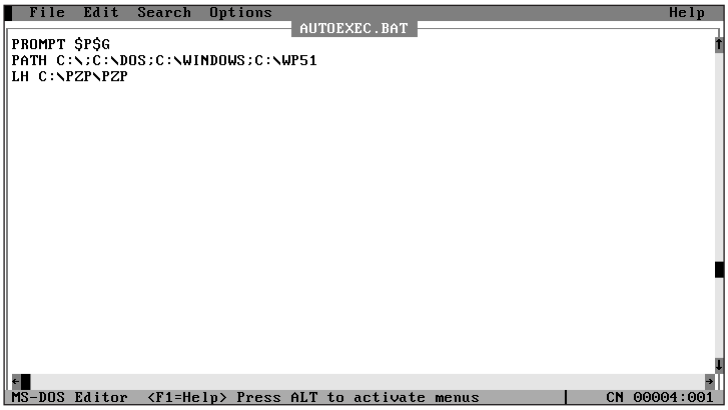


Figure 4-15 AUTOEXEC.BAT loading a TSR high

When TSRs are loaded high, two things can go wrong. Either the TSR might not work from upper memory, causing problems during execution, or there might not be enough room in upper memory for the program and its data. If the program causes the computer to hang when you attempt to run it, or if it simply refuses to work correctly, remove it from upper memory.

Simulating Expanded Memory

A+ OS 1.1

You can use HIMEM.SYS together with EMM386.EXE to simulate expanded memory. Some extended memory that is being managed by this memory management software can be allocated to "act" like expanded memory for software that requires it.

In Figure 4-16, the parameter 1024 RAM replaces the NOEMS parameter of Figure 4-14. By adding this parameter, you still have upper memory blocks available, but you also have told DOS to make 1024K of extended memory simulate expanded memory. Use this parameter when you are running software that can use expanded memory, instead of installing an expanded memory card. Consult the manual for the software that will use the expanded memory, in order to determine how much simulated expanded memory to allocate. Memory that simulates expanded memory is no longer available for extended memory applications.

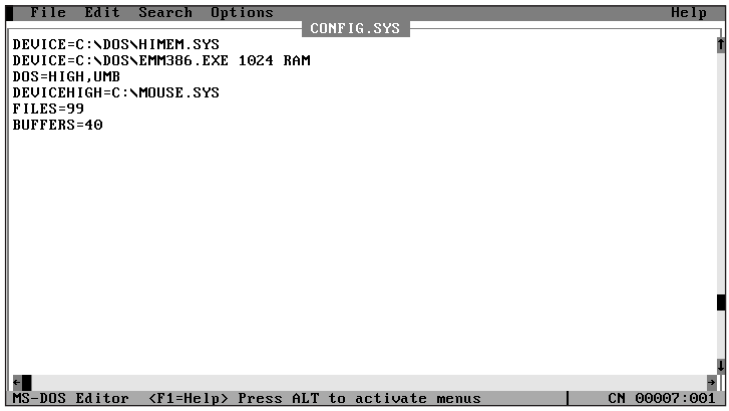


Figure 4-16 CONFIG.SYS set to simulate expanded memory

Memory Reports Using the MEM Command

A+ OS 3.1, 1.1

The DOS **MEM command** with the appropriate parameters shows exactly where in upper memory the UMBs are located, and what software has been assigned addresses in upper memory. In its simplest form, the MEM command looks like Figure 4-17. You will look at several examples of memory reports at different stages of memory management.

A+ OS 1.1, 3.1

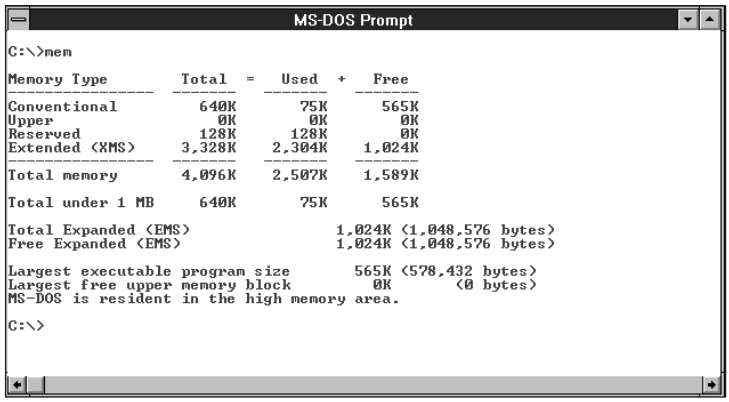


Figure 4-17 MEM report without UMBs available

In Figure 4-17, upper memory blocks are not being used, and the total memory under 1 MB is reported to be only 640K. In Figure 4-13, you saw the same PC with no changes to memory management. The MEM command in Figure 4-13 includes the /C option:

```
MEM /C |MORE
```

which shows a more complete report. You see several programs using conventional memory, and no use being made of the upper memory area.

To get a printed version of this report, use this command:

```
C:\> MEM/C >PRN
```

You can read the report one screen at a time using this command:

```
C:\> MEM/C x | MORE
```

In Figure 4-18, you can see how this upper memory area is helping to free some conventional memory after you have added the commands to CONFIG.SYS and AUTOEXEC.BAT to make use of upper memory. In Figure 4-19, the EMM386.EXE program was loaded from CONFIG.SYS, as in Figure 4-14, and two TSRs, SMARTDRV.EXE and SHARE.EXE, were loaded high from AUTOEXEC.BAT. Note that upper memory is now recognized by the system.

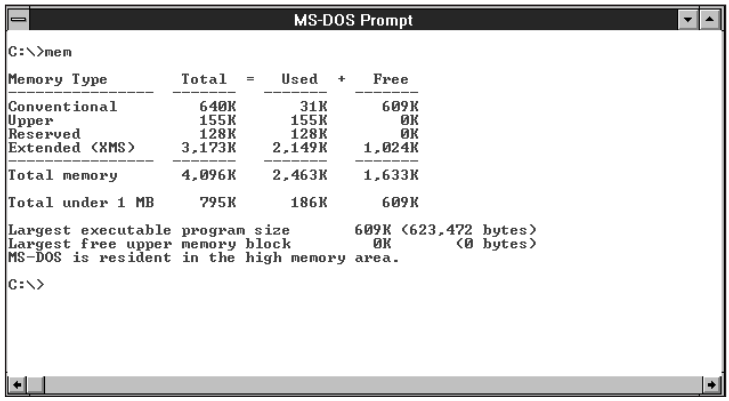


Figure 4-18 MEM report on a PC using upper memory

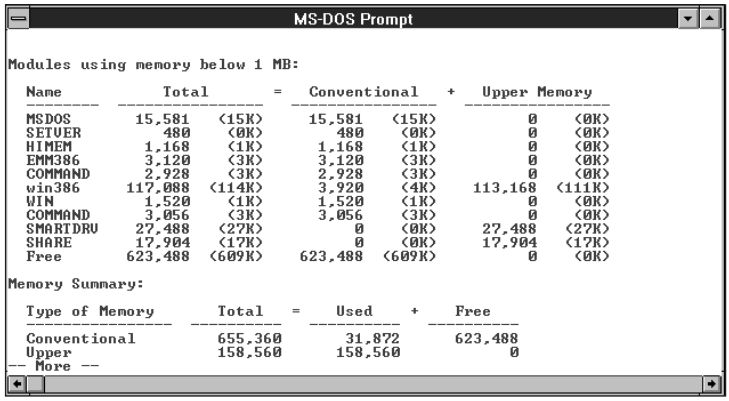


Figure 4-19 MEM/C report on a PC using upper memory

When you create upper memory blocks, you are reducing the total amount of extended memory available. This is because the amount of physical memory remains the same. What changes is the way DOS allocates memory addresses to this fixed amount of physical memory.

Compare Figure 4-13 with Figure 4-19. You see some interesting differences. EMM386 is requiring some memory in Figure 4-19, but was not used in Figure 4-13 since it had been omitted from CONFIG.SYS commands. Total conventional memory available for software increased from 578,448 in Figure 4-13 to 623,488 in Figure 4-19.

WIN386 is the Windows 3.x program responsible for managing virtual memory. Note the way WIN386 is using memory in both figures. In Figure 4-13, 6,704 bytes were being used when no upper memory blocks were available, but in Figure 4-19, WIN386 is using 117,088 bytes of memory. To see how this memory is being allocated, you use this command:

```
MEM /M WIN386
```

Figure 4-20 shows the results. Notice that all the upper memory area used by WIN386 is being used to store its data. WIN386 manages virtual memory for Windows and, in doing so, is able to decide to use upper memory area if it is available.

Using MemMaker with DOS 6+

DOS 6.0 and later versions offer an automatic way to manage upper and extended memory. MemMaker edits your CONFIG.SYS and AUTOEXEC.BAT files for you, placing the same commands in these files that were just discussed. It might edit an existing EMM386.EXE command line and change DEVICE= commands to DEVICEHIGH= commands. If it determines that a TSR loaded from AUTOEXEC.BAT can be loaded high, it adds the LH parameter to the beginning of the command line.

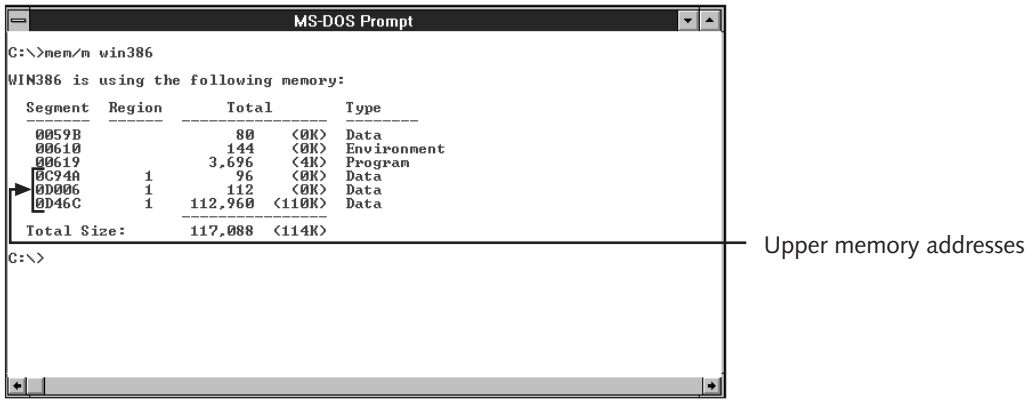


Figure 4-20 How WIN386 is using memory

To execute MemMaker, type **MEMMAKER** at the C prompt. MemMaker will ask you some questions about your system and then tell you that it is ready to reboot. During the boot process MemMaker monitors the loading of device drivers and other TSRs. After the boot process is complete, it makes some calculations to decide the best way to configure memory, and then edits the CONFIG.SYS and AUTOEXEC.BAT files. Sometimes it will edit the Windows System.ini file. Again, it will ask you to reboot the machine. After this reboot, it will ask if you saw any error messages during the boot process, and if your machine appears to be working well. If you say "Yes" to the latter question, then MemMaker is finished.

MemMaker saves the original files before it edits them. AUTOEXEC.BAT is saved as AUTOEXEC.UMB, CONFIG.SYS is saved as CONFIG.UMB, and System.ini is saved as SYSTEM.UMB. To return to these original files, type this command:

```
C:\> MEMMAKER /UNDO
```

The original copies of AUTOEXEC.BAT, CONFIG.SYS, and System.ini will be restored. Also, MemMaker offers a custom configuration. During this configuration, it will ask you what kind of video you are using. If you are not using a monochrome monitor, then it will use the B range of memory addresses normally designated for monochrome monitors for upper memory blocks.

If you have DOS 6+, you might want to compare your computer's performance before and after MemMaker has configured memory. Save the original CONFIG.SYS and AUTOEXEC.BAT files yourself, and compare the original and edited versions after MemMaker is finished. You might see some settings you might like to change in MemMaker's edited files; if so, you can make changes to the edited files yourself. It might take several iterations of MemMaker to completely optimize memory.

Managing Memory with Windows 9x

Memory management has not changed fundamentally with Windows 9x; memory is still organized as conventional, upper, extended, and expanded. However, Windows 9x has made some improvements in the allocation of this memory and in the automation of the process that make it easier for us to manage memory. One of the major improvements takes us to a new level of 32-bit protected mode drivers. These 32-bit drivers are automatically loaded into extended memory (not conventional or upper memory) when Windows 9x loads, thus eliminating the need for DEVICE= entries in CONFIG.SYS.

Software written with 32-bit code is generally faster and takes up more memory than older software written in 16-bit code. Windows 9x offers many drivers written in 32-bit code that can replace older 16-bit drivers written for DOS in real mode. New 32-bit drivers are sometimes called **virtual device drivers** or **VxD drivers**; they have .vxd or .386 file extensions and operate in protected mode. For Windows 9x to use older 16-bit drivers, it must provide a real-mode environment for these drivers to operate in, by using entries in CONFIG.SYS and AUTOEXEC.BAT that normally are not needed with Windows 9x. When given the choice, always choose 32-bit drivers over 16-bit drivers to increase overall computer performance.

Another improvement in Windows 9x is that it frees up more of conventional and upper memory, because it no longer uses SMARTDRV.EXE or SHARE.EXE, two TSRs that required a lot of memory below 1 MB. SMARTDRV.EXE, a 16-bit driver to manage disk caching, was replaced by Vcache, 32-bit disk caching that is built into Windows 9x. The 16-bit SHARE.EXE was replaced by the 32-bit Vshare.386, a part of Vmm32.exe that is automatically loaded when Windows 9x starts.

If you are fortunate enough to be using all 32-bit drivers and applications in a Windows 9x environment, memory management requires no work on your part. Just let Windows 9x automate the process for you. However, if you are running older 16-bit applications under Windows 9x, you might need to provide some of the same memory management aids needed by DOS. Windows 9x does not need AUTOEXEC.BAT or CONFIG.SYS files to run. However, if these files are present when Windows 9x boots, Windows 9x processes them just as DOS did.

Windows 9x uses HIMEM.SYS to manage extended memory just as DOS does, but, instead of being loaded from CONFIG.SYS, it's automatically loaded by Io.sys without requiring an entry in CONFIG.SYS. When using this automatic method, Io.sys does not load HIMEM.SYS until after the commands in CONFIG.SYS have been executed. If you are

using CONFIG.SYS to load EMM386.EXE, then you must include HIMEM.SYS in CONFIG.SYS along with EMM386.EXE because EMM386.EXE must find HIMEM.SYS already loaded before it can load.

If you need to load a 16-bit device driver into a UMB, then you must have a CONFIG.SYS file with these lines in it:

```
DEVICE=HIMEM.SYS
DEVICE=EMM386.EXE NOEMS
DOS=HIGH,UMB
```

However, if you are using all 32-bit drivers, you don't even need the CONFIG.SYS file. Windows 9x is much improved because it uses 32-bit drivers and software, but it is also backward compatible with 16-bit drivers and software that were written for DOS and Windows 3.x. When you are using these older programs, you have the same kinds of memory problems and require the same solutions as you did with DOS. All the commands for CONFIG.SYS and AUTOEXEC.BAT and the MEM command discussed earlier apply under Windows 9x using 16-bit drivers and older 16-bit software.

File extensions might help you to figure out whether you are using a 16-bit driver or a 32-bit driver to support a hardware device, but there is one way you can know for certain. Windows 3.x and DOS do not support 32-bit drivers, so if the driver works under DOS, you are using a 16-bit driver.

During the Windows 9x installation, Windows 9x setup tries to substitute 32-bit drivers for all 16-bit drivers it finds in use, and, if it can, to eliminate the CONFIG.SYS file altogether. However, if it can't substitute a 32-bit driver for the older 16-bit driver present, it puts (or keeps) the proper lines in the CONFIG.SYS file and sets itself up to use the older driver. More about this in Chapter 11.

When running DOS applications under Windows 9x, a DOS-like environment must be provided to the application. For example, to provide a DOS environment for the DOS program EDIT.COM on a PC running Windows 9x, go to Explorer and find the file EDIT.COM in the \Windows\Command directory. Right-click on the filename and select **Properties** from the drop-down menu that appears. Figure 4-21 is displayed, showing the Properties sheet for a DOS application. The entries on this sheet make up the **PIF** (**program information file**) for this application, which describes the environment the DOS program uses. Click on the **Memory** tab, and Figure 4-22 appears, listing the memory options available. From the Memory tab, you can specify how much conventional, expanded, and extended memory will be made available to the application, or leave the settings at Auto, which allows the application to use whatever is available. The last entry on the tab is MS-DOS protected-mode (DPMI) memory. This entry assigns the amount of protected-mode memory allowed the application. If you check Protected in the Conventional memory frame, the OS will protect memory used by the system from the application.

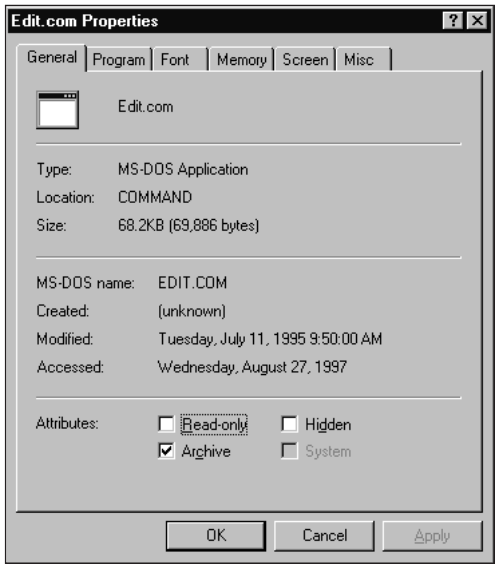


Figure 4-21 Properties sheet for a DOS application

Many DOS applications run with the default Auto settings with no problems. However, sometimes a DOS application has a problem with being given too much memory. Limit the amount of memory given the application using the Properties sheet shown in Figure 4-22.

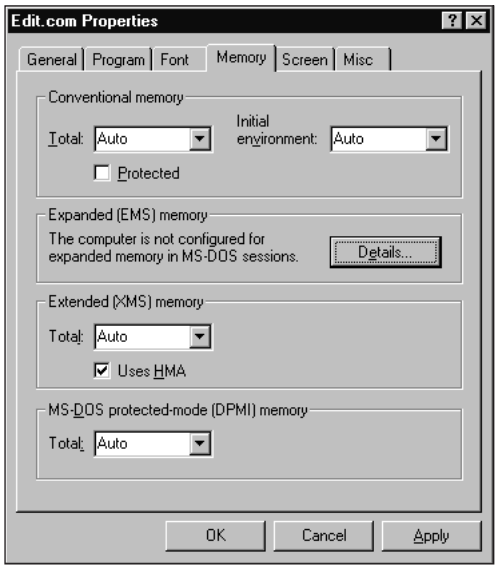


Figure 4-22 Setting up memory for a DOS application running under Windows 9x

Real Mode vs. Virtual Real Mode

When an OS that supports protected mode allows a 16-bit program that is written to work in real mode to run, this is called virtual real mode (sometimes referred to as virtual DOS mode). Figure 4-23 shows the difference between real mode and virtual real mode. In virtual real mode, the program "thinks" it is really working in a real-mode environment. It "thinks" that:

- It is the only program running
- It has all memory available to it, all 1024K of memory addresses that directly point to RAM
- It accesses data using a 16-bit data path

Underneath this environment, the OS is managing memory for the application. It receives the data in a 16-bit path, but is free to use a 32-bit data path to access memory and is also free to use virtual memory for the data.

Recall that there are two types of 16-bit applications: those written for DOS and those written for Windows 3.x. DOS 16-bit applications expect to run in real mode with no other applications running with them. A Windows 3.x 16-bit application expects to allow Windows to manage memory for it and expects that other applications might also be running in a cooperative multitasking environment.

When Windows 9x runs a 16-bit DOS application, ordinarily it is run in virtual DOS mode in a virtual DOS machine (VDM), sometimes called a DOS box, rather than in real mode. In a VDM, the application "thinks" it is running in real mode, but the OS is managing hardware resources using 32-bit drivers and providing virtual memory to the application. If you want a DOS application to have a "real" real mode rather than a virtual real mode, access the Properties box for the application. For example, once again using EDIT.COM as our example of a 16-bit application, right-click the program filename in Explorer (or you can right-click on the shortcut icon to the program) and select **Properties**. Click the **Program** tab and click **Advanced**. The Advanced Program Settings box is displayed as in Figure 4-24. Click **MS-DOS mode** to run the application in real mode. You can then choose to give the program its own private set of CONFIG.SYS and AUTOEXEC.BAT settings to be executed before the program runs. This information is stored in the PIF created for the shortcut. When you execute the shortcut or program, Windows 9x will shut down and reboot in real mode before executing the program.

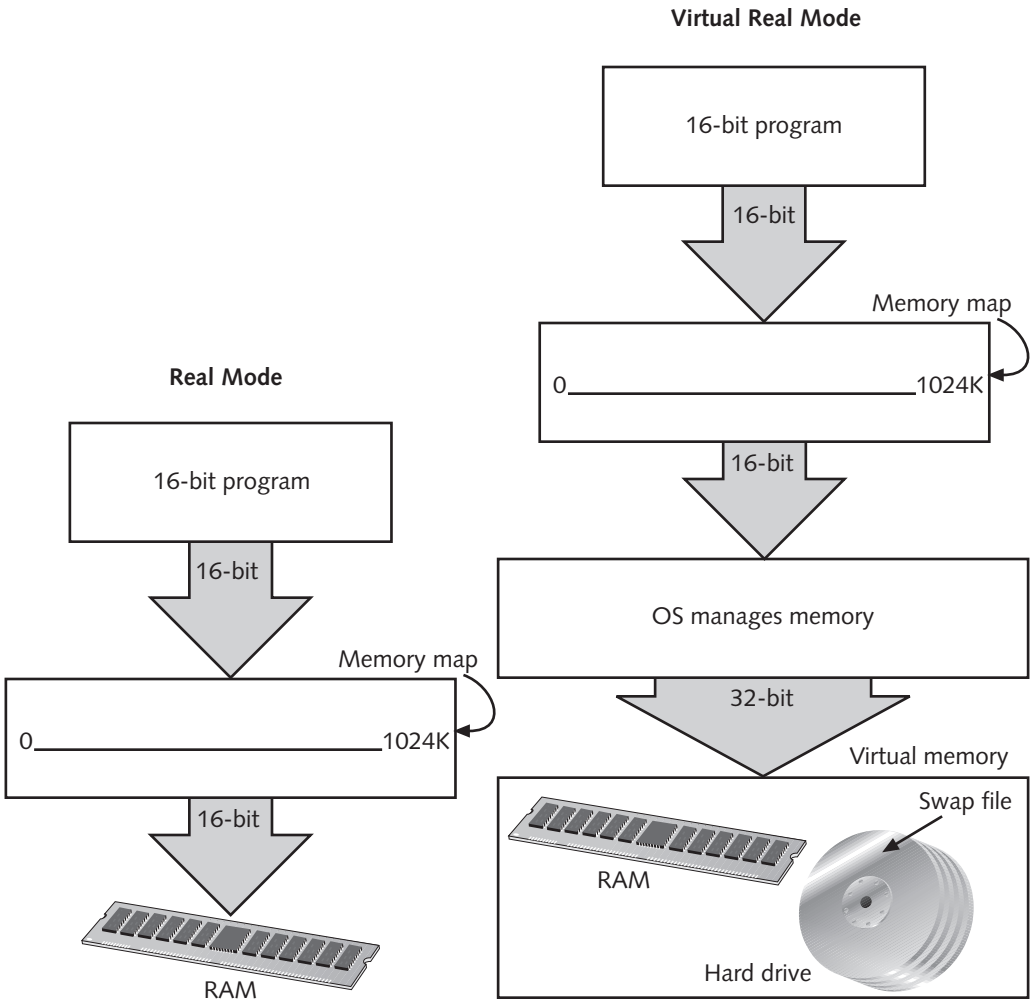


Figure 4-23 Virtual real mode provides "DOS in a box" to a 16-bit application that was written to run in real mode

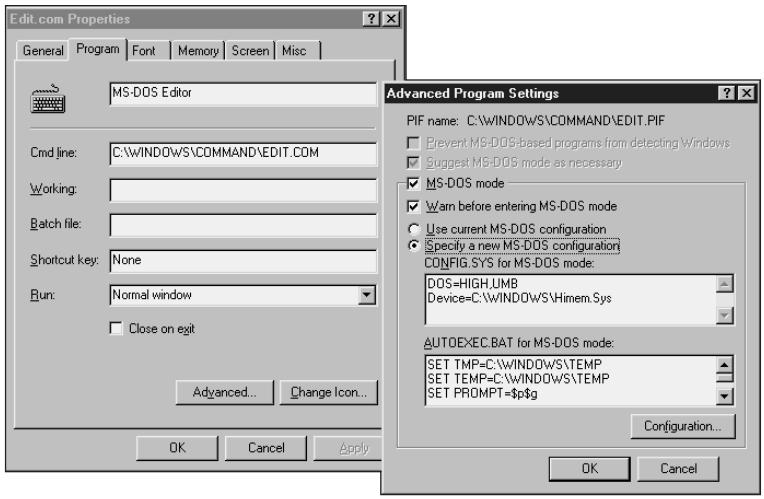


Figure 4-24 Use the Advanced Program Settings dialog box to run a 16-bit application in a real-mode environment in Windows 9x

To access real mode in Windows 9x, select **Start** and **Shutdown**. When the Shut Down Windows dialog box appears, select **Restart in MS-DOS mode**.

Windows 9x Swap File

A+ OS 3.1

Windows 9x automates the managing of virtual memory for you. Although you can override the automation, there is little reason to do so. To see what options Windows 9x offers, click **Start**, point to **Settings**, click **Control Panel**, then select **System** and select the **Performance** tab. A window similar to Figure 4-25 appears. Click **Virtual Memory**, and the dialog box in Figure 4-26 is displayed. Unless you have a very good reason to do otherwise, check **Let Windows manage my virtual memory settings**.

These settings are used to tell Windows how to manage the swap file. Notice in Figure 4-26 that you can specify the location of the swap file. You can choose to put the swap file on a compressed drive, but Windows does not compress the swap file itself in order to better ensure the safety of the file. Compressed drives, discussed in more detail in Chapter 6, are hard drives that have a portion of their data compressed in order to save space on the drive.

A+ OS 3.1

Figure 4-25 System Properties Performance box in Windows 9x

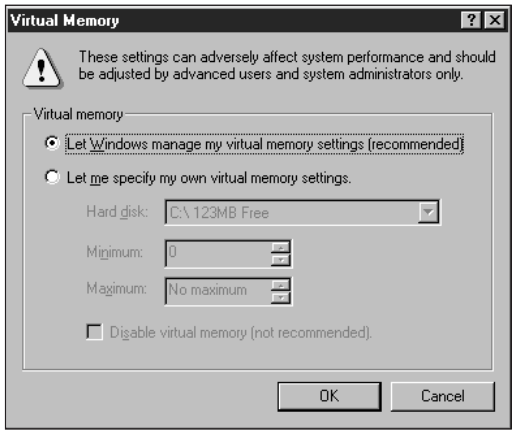


Figure 4-26 Options for managing virtual memory in Windows 9x

The Ultimate Solution: Windows NT and Windows 2000

A+ OS 1.1

Memory management under Windows NT and Windows 2000, compared to memory management under DOS and Windows 9x, is like traveling on a freeway rather than an obstacle course. This new approach, which began with Windows NT, also applies to Windows 2000. The memory mapping for Windows NT is one continuous, linear, 32-bit address space that allows each program and driver using Windows NT access to any part of this memory. Obstacles, out of the way!

A+ OS 1.1

Although it is not the intent of this chapter to delve too deeply into managing Windows NT memory, because of the complexity of dealing with conventional, upper, extended, and expanded memory, you can appreciate the ultimate solution to memory management on PCs—Windows NT.

The Windows NT memory management model is shown in Figure 4-27. Earlier in the chapter it was explained that expanded memory is accessed, by page frames, from upper memory and mapped onto the expanded memory card. The application thinks it has addresses in upper memory stored in RAM on the system board, when it really has only a window from upper memory mapped onto memory on the expansion board. You can compare this process to the memory management model for Windows NT.

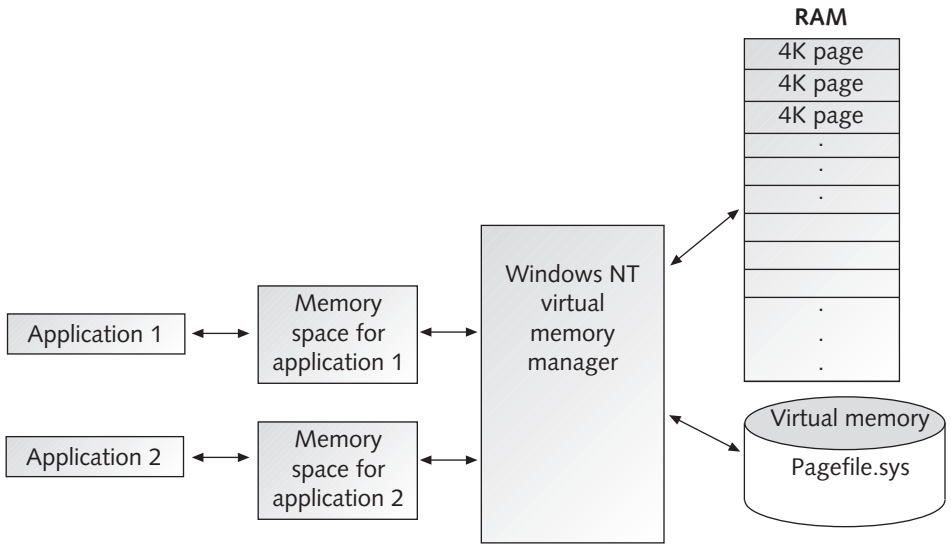


Figure 4-27 Windows NT and Windows 2000 memory management

Figure 4-27 shows the object-oriented approach to memory management. The application or device driver only says, "I want memory." It has no right to say to Windows NT which physical memory or which memory addresses it wants, or even the range of addresses that it wants to fall within. It can only say to Windows NT, "I want memory." Windows NT uses its Virtual Memory Manager to interface between the application or driver and the physical and virtual memory that it controls. Memory is allocated in 4K segments called **pages**. Applications and devices that are written for Windows NT only know how many pages they have. The virtual manager takes care of the rest. It is free to store these pages in RAM or on the hard drive in the swap file named Pagefile.sys. The only time an application would ever expect to run out of memory is when Pagefile.sys and RAM are full. Memory is only limited by the amount of physical memory available and the number of memory addresses that Windows NT can use, which is 4 GB.

With Windows NT and Windows 2000, struggles with "out of memory" errors should be a thing of the past.

Memory Management Troubleshooting Guidelines

A+ OS 3.2

This section describes what to do when things go wrong with memory management, and how to prevent potential errors.

When a TSR Will Not Load High Sometimes after instructing DOS to load a TSR high, a MEM report will show you that the load high instruction did not work. You included the DEVICE=HIMEM.SYS, the DEVICE=EMM386.EXE, and the DOS=UMB command lines in the CONFIG.SYS file, and you attempted to load a TSR high, using either the DEVICEHIGH= command in CONFIG.SYS or the LOADHIGH command in AUTOEXEC.BAT. But, when you get a MEM/C report, you discover that the TSR is still in conventional memory. Probably, the problem is either that the TSR did not have enough space in the UMB assigned to it, or that there were no more UMBs available for it. Do these things:

1. Check that all command lines in CONFIG.SYS and AUTOEXEC.BAT are correct.
2. Make sure that the DEVICEHIGH= commands in CONFIG.SYS are ordered so that the larger TSRs are placed first, in the larger UMBs. Still, they might not all fit in upper memory. By putting the largest ones possible in upper memory, you have freed up the largest amount of conventional memory possible.
3. MEMMAKER does a very good job of choosing the best order of loading TSRs. Try it.
4. Some TSRs do not work from upper memory, so test them before assuming that all is well.

When Devices Do Not Work or the System Hangs When DOS creates a UMB, DOS might "think" that no expansion board is using these memory addresses, when in fact they are being used by a card. If a TSR is loaded into this UMB, a memory conflict occurs. A memory conflict occurs when more than one program or their data have been assigned the same memory addresses. Memory conflicts can cause programs or devices to give errors, or can cause the computer to "lock up" and refuse to function.

There are two kinds of memory conflicts that cause a device to stop working or cause the system to hang:

1. Two expansion boards are using the same upper memory addresses.
2. DOS has created and is using a UMB in the same memory addresses used by an expansion board.

Two Expansion Boards Using the Same Upper Memory Addresses You have a scanner card installed to drive your scanner, and it works well. But when you install a network card in this computer, the scanner refuses to work, and the network does not work properly.

A+ OS 3.2

It is likely that both the network card and the scanner card are using the same upper memory addresses. Consider the following approaches:

1. Some expansion boards have DIP switches or jumpers that allow you to substitute one set of memory addresses for another.
2. You might change the memory addresses by adding a parameter to the command line that loads the device driver for the expansion board.
3. When two devices use the same memory addresses, you can often change one to an alternate set of memory addresses. If neither device accepts an alternate address, then you cannot install the two devices on the same computer. To find out which memory addresses are assigned to the devices, whether an alternate set of memory addresses is available, and how to access it, consult the documentation for the device or the software that drives the device. It is very important to keep the hardware and software documentation in a place where you can find it. Without the documentation, your only recourse is to call technical support for the software or hardware.

When reading the documentation, you will find that most addresses are given in hex rather than decimal form. (See Appendix D for an explanation of the hex number system.) Sometimes the memory addresses are written without the last hexadecimal numeral. For example, if the documentation says that the device uses C800 through CFFF, interpret this to mean that the upper memory address range is C8000 through CFFFF. Once you have discovered that the two devices use the same memory addresses, find out if one can use alternate addresses. If so, your problem is solved.

A+ OS 3.2

When UMBs and Expansion Boards Conflict When DOS creates a UMB, it assigns memory addresses that it "thinks" are not being used by devices. However, some devices don't tell DOS what memory addresses they are using until the device is activated after booting. This delay causes DOS to think that the memory addresses assigned to a device are available, and DOS creates and loads a TSR into the UMB. If this conflict happens, the system might hang, the TSR might not work properly, and/or the device might not work properly. Try the following approach:

1. Read the documentation that came with the device to find out which memory addresses it is using. Some device drivers display the memory addresses they are using during the boot process as they are loaded. Carefully watch for this information on the screen while booting. Also, try using MSD to display how memory is being used. Once you know the memory addresses being used, you can change the EMM386.EXE command line so that this range of addresses is not used.
2. Use the Exclude option to the EMM386.EXE command line to exclude certain memory addresses. Do not use the last numeral in the hex address in the command line. For example, suppose you read from the documentation that came with the device that it uses addresses CC000–CFFFF. To exclude these addresses from the addresses used by UMBs, use this command line:

```
DEVICE=C:\DOS\EMM386.EXE NOEMS X = CC00 - CFFF
```

3. Reboot your computer to activate the change. The memory conflict problem should then be solved.

Upgrading Memory

A+CORE 1.8, 4.2

Upgrading memory means to add more RAM to a computer. Many computers, when first purchased, have empty slots on the system board, allowing you to add SIMMs or DIMMs to increase the amount of RAM. If all the slots are full, sometimes you can take out small-capacity modules and replace them with larger-capacity modules. When you add more memory to your computer, ask yourself these questions:

- How much memory do I need?
- How much memory can my computer physically accommodate?
- What increments of memory does my system board support?
- How much additional memory is cost effective?
- What kind of memory can fit on my system board?
- What memory is compatible with the memory I already have installed?

With the demands today's software places on memory, the answer to the first question is probably, "All I can get." Both Windows 95 and Windows 98 need from 24 MB to 32 MB of memory. The minimum requirement is 8 MB, although performance will be slow with this little memory because the system is forced to write working files to the slower hard drive as virtual memory instead of using the much faster RAM.

How Much Memory Can Fit on the System Board?

To determine how much memory your computer can physically hold, read the documentation that comes with your computer. For example, one manual for a 486 computer explains that the system board can support up to 32 MB of memory, but there are only nine possible memory configurations. A table that describes these configurations might look like Table 4-4.

Table 4-4 shows that this system board has two banks. A **bank** is a location on the system board that contains slots for memory modules. On this system board, each bank can hold 256K, 1 MB, or 4 MB of memory. The first bank always has some memory in it, but the second bank might or might not contain memory. This computer, which is typical of many older 486 boards, uses SIMMs on the system board. Recall that a SIMM is a small miniboard that contains memory chips. The SIMMs are inserted into the slots in a bank. This computer can support these sizes: 256K, 1 MB, and 4 MB.

A+CORE 4.2

Table 4-4 Memory configurations of a 486 system board

SIMM Size in Bank 1	SIMM Size in Bank 2	Total RAM on System Board
256K	0	1 MB
256K	256K	2 MB
1 MB	0	4 MB
1 MB	256K	5 MB
1 MB	1 MB	8 MB
4 MB	0	16 MB
4 MB	256K	17 MB
4 MB	1 MB	20 MB
4 MB	4 MB	32 MB

To determine how many slots are in one bank, you must know the computer's bus size. This 486 computer uses a 32-bit bus. When bits travel down a circuit on the system board to the bank to be stored in RAM, they are moving 32 bits abreast. The bank must receive 32 bits at a time to work with this bus. This system board uses 30-pin SIMMs. Each 30-pin SIMM receives one 8-bit byte at a time. Figure 4-28 shows that the 32-bit bus directs 8 bits to each of four SIMMs in the bank. The bank must contain four SIMMs to receive these 32 bits. Since each SIMM receives an equal part of the 4 bytes traveling down the circuit, all SIMMs within one bank must store the same amount of bytes.

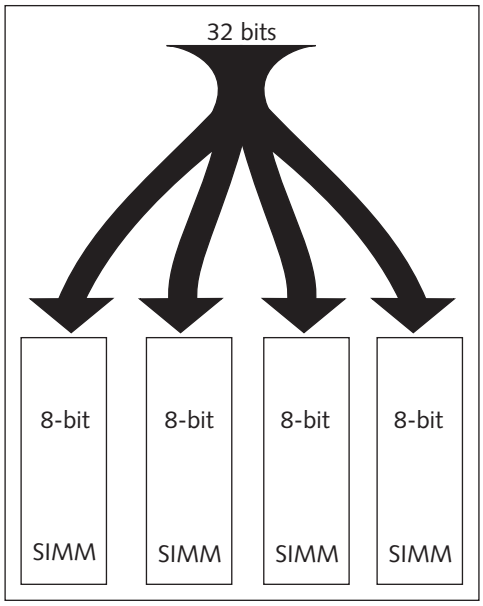


Figure 4-28 One bank on a 486 system board that uses a 32-bit bus and 8-bit, 30-pin SIMMs; each SIMM must hold the same amount of memory

A+CORE 4.2

In Table 4-4, you see in the first row of the table that bank 1 contains 256K SIMMs, and bank 2 is empty. Bank 1 contains four slots, each of which must contain a SIMM in order to accommodate the 32-bit bus. Hence, the amount of memory is 4 × 256K or 1 MB of memory. In the second row of the table, each bank contains four 256K SIMMs for a total of 2 MB of memory: (4 × 256K) + (4 × 256K) = 2 MB.

Notice that in the fourth row of the table, bank 1 contains 1-MB SIMMs, and bank 2 contains 256K SIMMs for a total of 5 MB of memory on the system board (calculations are left to the reader). The four SIMMs in a bank must be the same size, but the SIMMs can vary in size from one bank to another.

Our second example of a system board is one used by a Pentium, and it supports up to 128 MB of RAM. It has four SIMM sockets divided into two banks. Bank 0 holds SIMMs 1 and 2, and bank 1 holds SIMMs 3 and 4 (see Figure 4-29). Memory can be installed using 4-MB, 8-MB, and 16-MB SIMMs using either 72-pin EDO or FPM modules, which must have at least 70 ns speed.

Remember from Chapter 3 that the Pentium memory bus between RAM and the CPU is 64 bits wide (see Figure 3-23). Most 72-pin SIMM modules sold today accommodate a 32-bit data path so two SIMMs must be paired together to receive data from the 64-bit Pentium memory bus. One bank of memory on the Pentium system board must contain two 32-bit SIMMs. The other bank does not need to be filled, but, if it is used, both of the two SIMMs must be present in that second bank. As you study Figure 4-29, remember that this is the Pentium memory bus connecting the CPU and RAM, not the PCI bus, which is only 32 bits wide.

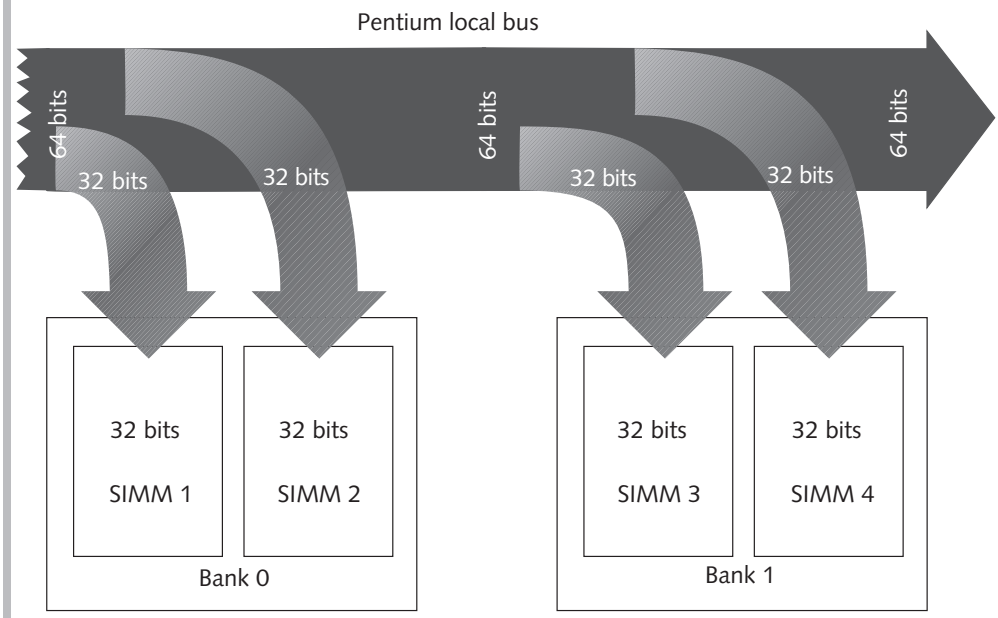


Figure 4-29 A Pentium memory bus is 64 bits wide and requires two 32-bit SIMMs to accommodate the bus width; each 64-bit bank can be used independently of the other

A+CORE 4.2

Table 4-5 shows half of the memory configurations supported by a Pentium system board using SIMMs. The other half is in a project at the end of this chapter with calculations left to the reader.

Pentium system boards that use DIMM modules use only one socket to a bank, since a DIMM module accommodates a data path of 64 bits. Single-sided modules (chips on only one side of the module) come in 8, 16, 32, 64, and 128 MB sizes, and double-sided modules (chips on both sides) come in 32, 64, 128, and 256 MB sizes. Figure 4-30 shows how one, two, or three sockets of DIMMs can be used by a Pentium system board.

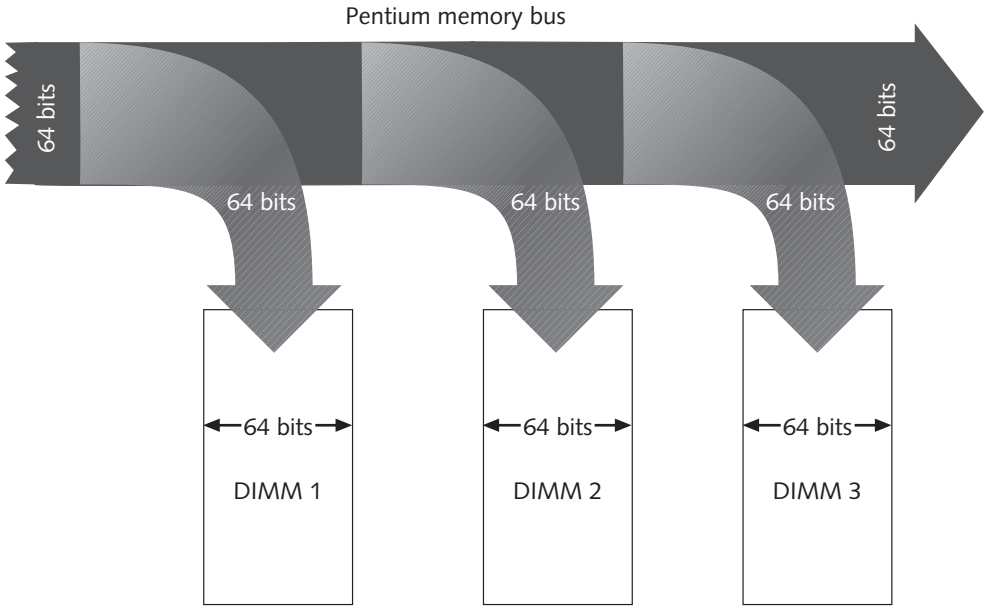


Figure 4-30 Only a single DIMM is needed to fill one bank of memory for a Pentium memory bus; each DIMM can be a different size

Table 4-5 Memory configurations for a Pentium system board using SIMMs

SIMM Size in Bank 0	SIMM Size in Bank 1	Total Memory
4 MB	0	8 MB
4 MB	4 MB	16 MB
4 MB	8 MB	24 MB
4 MB	16 MB	40 MB
4 MB	32 MB	72 MB
8 MB	0	16 MB
8 MB	4 MB	24 MB
8 MB	8 MB	32 MB

Table 4-5 Memory configurations for a Pentium system board using SIMMs (continued)

SIMM Size in Bank 0	SIMM Size in Bank 1	Total Memory
8 MB	16 MB	48 MB
8 MB	32 MB	80 MB
16 MB	0	32 MB
16 MB	4 MB	40 MB
16 MB	8 MB	48 MB

Selecting Memory Types

When you are placing memory on the system board, match the type of memory to the system board requirements. For example, for the first Pentium system board just discussed, the documentation says that you must use 72-pin SIMMs, which can be either EDO or FPM modules. The speed must be at least 70 ns. Avoid mixing speeds on the same system board. If you use a SIMM having one speed in one bank and a SIMM having another speed in the other bank, your computer will only work as fast as the slower bank. Always put the slower SIMMs in the first bank. However, to ensure the most reliable results, use the same speed of SIMMs in all banks and also buy the same brand of SIMMs.

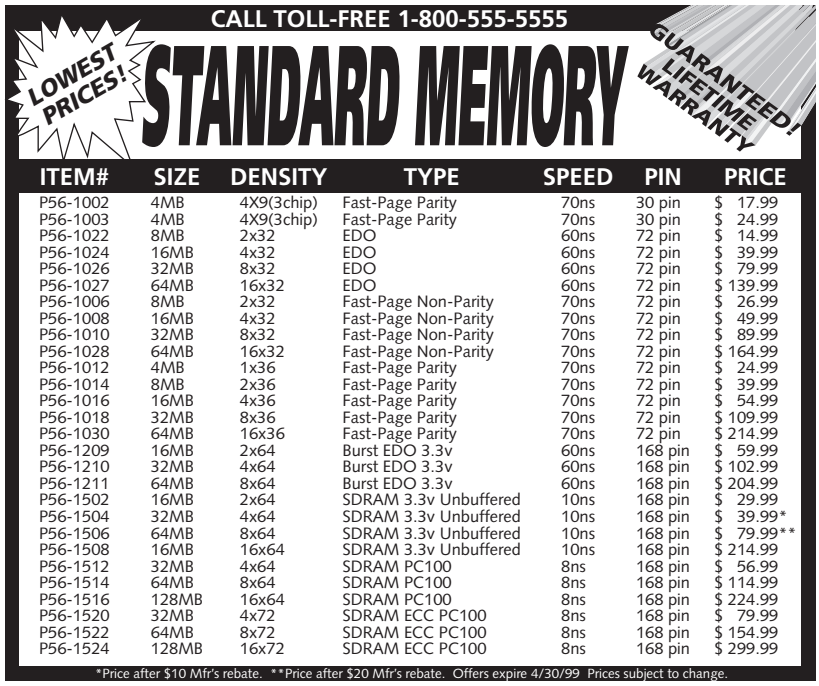
For the second Pentium system board just discussed, which uses 168-pin DIMM modules, the documentation says to use unbuffered, 3.3V, PC100 DIMM SDRAM modules. The PC100 refers to the speed of the modules, meaning that the modules should be rated to work with a system board that runs at 100 MHz. You have the choice of using ECC modules. If you choose to not use them, then CMOS setup should show the feature disabled. There are three DIMM sockets on the board, and each socket represents one bank. Figure 4-31 shows the possible combinations of DIMMS that can be installed in these sockets.

DIMM Location	168-pin DIMM		Total Memory
Socket 1 (Rows 0&1)	SDRAM 8, 16, 32, 64, 128, 256MB	x1	
Socket 2 (Rows 2&3)	SDRAM 8, 16, 32, 64, 128, 256MB	x1	
Socket 3 (Rows 4&5)	SDRAM 8, 16, 32, 64, 128, 256MB	x1	
	Total System Memory (Max 768MB)	=	

Figure 4-31 This table is part of the system board documentation and is used to show possible DIMM sizes and calculate total memory on the system board

Reading Ads About Memory Modules

Figure 4-32 shows a typical memory module ad listing 30-pin SIMMs, 72-pin EDO and FPM SIMMs, and BEDO and SDRAM DIMMs. When you are selecting memory, the number of pins, the speed, the size, and the type of module are all important. We will now examine the ad to see how all this information is given to us. The second column in the ad shows the total amount of memory on each module. For example, the first entry shows a 4-MB 30-pin SIMM.



CALL TOLL-FREE 1-800-555-5555

LOWEST PRICES!

STANDARD MEMORY

GUARANTEED! LIFETIME WARRANTY

ITEM#	SIZE	DENSITY	TYPE	SPEED	PIN	PRICE
P56-1002	4MB	4X9(3chip)	Fast-Page Parity	70ns	30 pin	$ 17.99
P56-1003	4MB	4X9(3chip)	Fast-Page Parity	70ns	30 pin	$ 24.99
P56-1022	8MB	2x32	EDO	60ns	72 pin	$ 14.99
P56-1024	16MB	4x32	EDO	60ns	72 pin	$ 39.99
P56-1026	32MB	8x32	EDO	60ns	72 pin	$ 79.99
P56-1027	64MB	16x32	EDO	60ns	72 pin	$ 139.99
P56-1006	8MB	2x32	Fast-Page Non-Parity	70ns	72 pin	$ 26.99
P56-1008	16MB	4x32	Fast-Page Non-Parity	70ns	72 pin	$ 49.99
P56-1010	32MB	8x32	Fast-Page Non-Parity	70ns	72 pin	$ 89.99
P56-1028	64MB	16x32	Fast-Page Non-Parity	70ns	72 pin	$ 164.99
P56-1012	4MB	1x36	Fast-Page Parity	70ns	72 pin	$ 24.99
P56-1014	8MB	2x36	Fast-Page Parity	70ns	72 pin	$ 39.99
P56-1016	16MB	4x36	Fast-Page Parity	70ns	72 pin	$ 54.99
P56-1018	32MB	8x36	Fast-Page Parity	70ns	72 pin	$ 109.99
P56-1030	64MB	16x36	Fast-Page Parity	70ns	72 pin	$ 214.99
P56-1209	16MB	2x64	Burst EDO 3.3v	60ns	168 pin	$ 59.99
P56-1210	32MB	4x64	Burst EDO 3.3v	60ns	168 pin	$ 102.99
P56-1211	64MB	8x64	Burst EDO 3.3v	60ns	168 pin	$ 204.99
P56-1502	16MB	2x64	SDRAM 3.3v Unbuffered	10ns	168 pin	$ 29.99
P56-1504	32MB	4x64	SDRAM 3.3v Unbuffered	10ns	168 pin	$ 39.99*
P56-1506	64MB	8x64	SDRAM 3.3v Unbuffered	10ns	168 pin	$ 79.99**
P56-1508	16MB	16x64	SDRAM 3.3v Unbuffered	10ns	168 pin	$ 214.99
P56-1512	32MB	4x64	SDRAM PC100	8ns	168 pin	$ 56.99
P56-1514	64MB	8x64	SDRAM PC100	8ns	168 pin	$ 114.99
P56-1516	128MB	16x64	SDRAM PC100	8ns	168 pin	$ 224.99
P56-1520	32MB	4x72	SDRAM ECC PC100	8ns	168 pin	$ 79.99
P56-1522	64MB	8x72	SDRAM ECC PC100	8ns	168 pin	$ 154.99
P56-1524	128MB	16x72	SDRAM ECC PC100	8ns	168 pin	$ 299.99

*Price after $10 Mfr's rebate. **Price after $20 Mfr's rebate. Offers expire 4/30/99 Prices subject to change.

Figure 4-32 Typical ad for memory modules

The third column gives the density of the module, which tells us the width of the data bus, if the module supports error checking, and the size of the module. Here's how it works. The density is written as two numbers separated by an $\times$, such as 8×32, and is read "8 by 32." Let's start with the second number. If the second number is 32 or 64, then it's the width of the data bus in bits grouped as 8 bits to a byte. If the number is 36 or 72, then it's the width of the data bus plus an extra bit for each byte that is used for either parity (for parity memory) or error checking and correction (for ECC memory). Look down the third column and note where you see 36 or 72, to verify that these are the values listed for parity or ECC memory. When calculating the size of the module, ignore the ninth bit and use only the values 32 or 64 to calculate size. Convert this number to bytes by dividing the number by 8, and then multiply that value (number of bytes) by the number on the left in the density listing to get the size of the module. For example, if the density is 8×32, then the size of the module is $8 \times (32/8) = 8 \times 4 = 32$ MB.

Here is another example: if the density is 16×36, then ignore the parity bit and use 16×32 to do your calculations, giving $16 \times 32 = 16 \times (32/8) = 16 \times 4 = 64$ MB. The size of the module is 64 MB; the data path is 32 bits, and the module does support parity.

Notice in the fifth column of the ad the big difference between the speed of SDRAM and that of other types of memory. Also notice the difference in price between SDRAM for a 100-MHz system board that has ECC and memory that does not have ECC functionality.

Installing Memory

A+CORE 1.2

When installing SIMMs or DIMMs, remember to protect the chips against static electricity. Always use a ground bracelet as you work. Turn the power off and remove the cover to the case. Handle memory modules with care. Ground yourself before unpacking or picking up a card. Don't stack cards, because you can loosen a chip. Usually modules pop easily into place and are secured by spring catches on both ends. Look for the notch on one side of the SIMM module that orients the module in the slot. The module slides into the slot at an angle, as shown in Figure 4-33. Place each module securely in its slot. Turn on the PC and watch the amount of memory being counted by POST during the boot process. If all the memory you expect does not count up correctly, remove and reseat each module carefully. To remove a module, release the latches on both sides of the module and gently rotate the module out of the socket at a 45-degree angle.

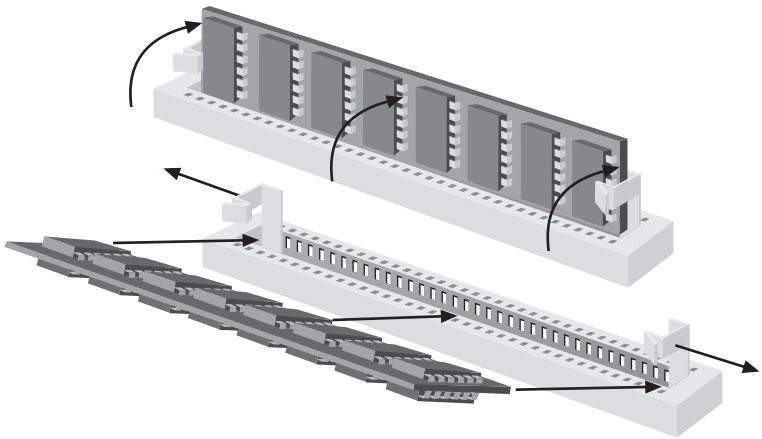


Figure 4-33 Installing a SIMM module

For DIMM modules, as shown in Figure 4-34, small latches on either side of the slot hold the module in place. Look for the notches on the DIMM module to orient it into the slot. Insert the module straight down into the slot, just as you would an expansion card.

For RIMM modules, the technology works so that the signal enters one end of the first RIMM socket and out the other end and then on to the next socket. For this reason, each socket must be filled so that continuity throughout all sockets is maintained. If the socket does not hold a RIMM, then it must hold a placeholder module called a C-RIMM (Continuity RIMM) to assure continuity throughout all slots. The C-RIMM does not contain any memory.

A+CORE 1.2

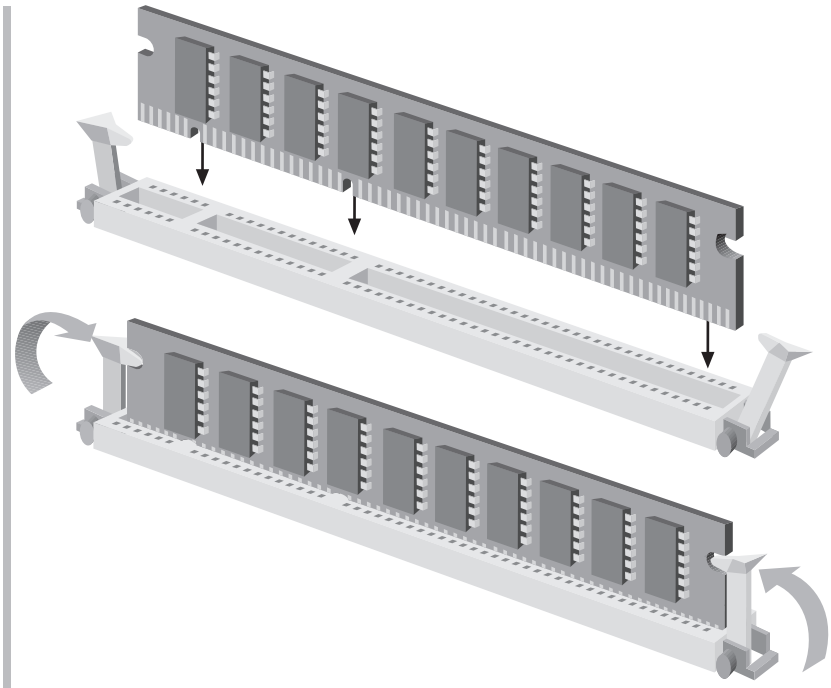


Figure 4-34 Installing a DIMM module

Most often, placing memory on the system board is all that is necessary for installation. When the computer powers up, it counts the memory present without any further instruction and senses what features the modules support, such as parity or ECC. For some computers, you must tell the setup how much memory is present. Read the instructions that come with your computer to determine what yours requires.

CHAPTER SUMMARY

- Memory can be viewed as both physical memory installed on the system board and circuit boards and as logical memory managed by the OS.
- The two kinds of physical memory are RAM and ROM.
- In order for ROM or RAM physical memory to be used by the computer, memory addresses must be assigned to it.
- System BIOS is stored on a ROM chip on the system board. In addition, expansion boards sometimes have ROM chips on them, holding BIOS programming to manage a device.
- The CPU uses memory in two ways, as main memory and as a memory cache.

- SRAM is fast, static RAM and is used as a memory cache, which speeds up the overall computer performance by temporarily holding data and programming that may possibly be used by the CPU in the near future. SRAM does not require constant refreshing.
- DRAM (dynamic RAM) is slower than SRAM because it needs constant refreshing.
- DRAM is stored on two kinds of miniboards: SIMMs and DIMMs.
- SIMM memory modules can use either EDO or FPM technology. EDO is faster and only slightly more expensive than FPM, but the system board must support this type of memory to make use of its increased speed.
- DIMM memory modules can use either BEDO or synchronous DRAM (SDRAM).
- Direct Rambus DRAM and Double Data Rate SDRAM (DDR SDRAM) are two technologies that are contending to be the next DRAM technology standard.
- Flash memory holds data permanently until it is overwritten, and is commonly used on Flash ROM chips and memory cards for notebook computers.
- Synchronous DRAM (which moves in sync with the memory bus) is a faster kind of memory than the less expensive asynchronous DRAM (which does not move in sync with the memory bus) found on SIMMs.
- When buying memory, use only gold edge connectors on memory modules that will be inserted in slots containing gold connections, and use only tin connectors in tin slots.
- When buying memory, beware of remanufactured and re-marked memory chips because they have been either refurbished or re-marked before resale.
- SRAM comes as either synchronous or asynchronous memory. Synchronous is faster and slightly more expensive than asynchronous memory.
- Synchronous SRAM can come as either burst or pipelined burst memory.
- COAST is a cache memory module holding pipelined burst SRAM chips.
- Logical memory is divided into conventional memory, upper memory, and extended memory, according to the memory addresses assigned to it.
- Upper memory is traditionally used to hold BIOS and device drivers. Drivers for legacy video cards normally fill the A, B, and C range of upper memory addresses (hex addresses beginning with A, B, and C).
- The beginning of extended memory is called the high memory area and can hold a portion of DOS.
- Expanded memory is located on an expansion board and is accessed by page frames given upper memory addresses.
- Windows can emulate expanded memory, taking some of RAM and presenting it to applications software as expanded memory.

- The practice of copying BIOS from slower ROM chips to faster RAM chips for processing is called shadowing ROM. The area of RAM holding the BIOS is called shadow RAM.
- Virtual memory is space on the hard drive that is used by the OS as pseudo-memory.
- A RAM drive is space in memory that is used as a pseudo-hard-drive.
- DOS and Windows 9x use the device driver HIMEM.SYS to manage extended memory.
- DOS uses EMM386.EXE to make more efficient use of upper memory addresses and to emulate expanded memory.
- An upper memory block (UMB) is a group of upper memory addresses made available to TSRs.
- Storing device drivers and TSRs in upper memory is called loading high.
- DOS can load device drivers into upper memory blocks by using the DEVICEHIGH command in CONFIG.SYS.
- DOS can load a TSR high by using the LOADHIGH command in AUTOEXEC.BAT.
- MemMaker is a DOS utility that can help in managing upper memory addresses.
- A swap file is the file on the hard drive that is used by the OS as virtual memory.
- Windows NT and Windows 2000 use an approach to memory management that is altogether different from that of DOS and Windows 9x. Conventional, upper, and extended memory concepts do not exist in Windows NT or Windows 2000.
- Memory modules must be installed on a system board in the slots of a memory bank according to the rules specified in the system-board documentation. There are a fixed number of memory configurations that a board supports.

KEY TERMS

Asynchronous SRAM — Static RAM that does not work in step with the CPU clock and is, therefore, slower than synchronous SRAM.

Bank — An area on the system board that contains slots for memory modules (typically labeled bank 0, 1, 2, and 3).

Base memory — See conventional memory.

Buffer — A temporary memory area where data is kept before being written to a drive or sent to a printer, thus reducing the number of writes needed when devices communicate at different speeds.

Burst EDO (BEDO) — A refined version of EDO memory that significantly improved access time over EDO. BEDO is not widely used today because Intel chose not to support it. BEDO memory is stored on 168-pin DIMM modules.

Burst SRAM — Memory that is more expensive and slightly faster than pipelined burst SRAM. Data is sent as a two-step process; the data address is sent, and then the data itself is sent without interruption.

COAST (cache on a stick) — Chips on a module available for pipelined burst synchronous SRAM.

Conventional memory — Memory addresses between 0 and 640K. Also called base memory.

Double-data rate SDRAM (DDR SDRAM or **SDRAM II)** — A type of memory technology used on DIMMs that runs at twice the speed of the system clock.

Direct Rambus DRAM — A memory technology by Rambus and Intel that uses a narrow, very fast network-type memory bus. Memory is stored on a RIMM module. Also called **RDRAM** or **Direct RDRAM**.

Dynamic RAM (DRAM) — Common system memory with access speeds ranging from 70 to 50 nanoseconds, requiring refreshing every few milliseconds.

ECC (error checking and correction) — A chip set feature on a system board that checks the integrity of data stored on DIMMs and can correct single-bit errors in a byte. More advanced ECC schemas can detect, but not correct, double-bit errors in a byte.

EDO (extended data output) memory — A type of RAM that may be 10-20% faster than conventional RAM because it eliminates the delay before it issues the next memory address.

EEPROM (electrically erasable programmable ROM) chip — A type of chip in which higher voltage may be applied to one of the pins to erase its previous memory before a new instruction set is electronically written.

EMM386.EXE — A DOS utility that provides both emulated expanded memory (EMS) and upper memory blocks (UMBs).

EPROM (erasable programmable ROM) chip — A type of chip with a special window that allows the current memory contents to be erased with special ultraviolet light so that the chip can be reprogrammed. Many BIOS chips are EPROMs.

Expanded memory (EMS) — Memory outside of the conventional 640K and the extended 1024K range that is accessed in 16K segments, or pages, by way of a window to upper memory.

Extended memory — Memory above the initial 1024 KB, or 1 MB, area.

Flash memory — A type of RAM that can electronically hold memory even when the power is off.

FPM (fast page mode) memory — An earlier memory mode used before the introduction of EDO memory.

General Protection Fault (GPF) error — A Windows error that occurs when a program attempts to access a memory address that is not available or is no longer assigned to it.

High memory area (HMA) — The first 64K of extended memory. The method of storing part of DOS in the high memory area is called loading DOS high.

HIMEM.SYS — A device driver that manages memory above 640K. It is often executed by the line DEVICE = C:\DOS\HIMEM.SYS in a CONFIG.SYS file.

Load size — The largest amount of memory that a driver needs to initialize itself and to hold its data. It is almost always a little larger than the size of the program file.

Loading high — The process of loading a driver or TSR into upper memory.

MEM command — A DOS utility used to display how programs and drivers are using conventional, upper, and extended memory (Example: MEM/C/P).

MemMaker — A DOS utility that can increase the amount of conventional memory available to DOS-based software applications, by loading drivers and TSRs into upper memory.

Memory — Physical microchips that can hold data and programming located on the system board or expansion cards.

Memory address — A number that the CPU assigns to physical memory to keep track of the memory that it has access to.

Memory caching — Using a small amount of faster RAM to store recently retrieved data, in anticipation of what the CPU will next request, thus speeding up access.

Memory management — The process of increasing available conventional memory, required by DOS-based programs, accomplished by loading device drivers and TSRs into upper memory.

Memory mapping — Assigning addresses to both RAM and ROM during the boot process.

Nonparity memory — Slightly less expensive, 8-bit memory without error checking. A SIMM part number with a 32 in it (4 × 8 bits) is nonparity.

Page — Memory allocated in 4K or 16K segments within a page frame.

Page frame — A 64K upper memory area divided into four equal-sized pages through which the memory manager swaps data.

Parity memory — Nine-bit memory in which the 9th bit is used for error checking. A SIMM part number with a 36 in it (4 × 9 bits) is parity. Older DOS PCs almost always use parity chips.

Pipelined burst SRAM — A less expensive SRAM that uses more clock cycles per transfer than nonpipelined burst, but does not significantly slow down the process.

Program Information File (PIF) — A file used by Windows to describe the environment for a DOS program to use.

RAM drive — A RAM area configured as a virtual hard drive, such as drive D, so that frequently used programs can be accessed faster. It is the opposite of virtual memory.

Re-marked chips — Chips that have been used and returned to the factory, marked again, and resold. The surface of the chips may be dull or scratched.

Shadow RAM or **shadowing ROM** — The process of copying ROM programming code into RAM to speed up the system operation, because of the faster access speed of RAM.

Static RAM (SRAM) — RAM chips that retain information without the need for refreshing as long as the power is on. They are more expensive but less volatile than traditional DRAM.

Swapping — A method of freeing some memory by moving a "page" of data temporarily to a swap file on the hard drive; it can later be copied from disk back into memory.

Synchronous DRAM (SDRAM) — A type of memory stored on DIMMs that run in sync with the system clock, running at the same speed as the system board. Currently, the fastest memory used on PCs.

Synchronous SRAM — SRAM that is faster and more expensive than asynchronous SRAM. It requires a clock signal to validate its control signals, enabling the cache to run in step with the CPU.

SyncLink (SLDRAM) — A synchronous memory technology that increases the number of memory banks from 4 to 16.

System variable — A variable that has been given a name and a value; it is available to the operating system and applications software programs.

Temp directory — A location to which inactive applications and data can be moved as a swap file, while Windows continues to process current active applications. (Avoid deleting Temp swap while Windows is running.)

Temporary file — A file that is created by Windows applications, to save temporary data, and may or may not be deleted when the application is unloaded.

Upper memory — The memory addresses from 640K up to 1024K, originally reserved for BIOS, device drivers, and TSRs.

Upper memory block (UMB) — A group of consecutive memory addresses in RAM from 640K to 1 MB that can be used by device drivers and TSRs.

Virtual device driver (VDD) or **VxD driver** — A 32-bit device driver running in protected mode.

Virtual memory — Hard disk space used when a system starts to run low on RAM. Because hard drives are much slower than RAM access, virtual memory is relatively slow.

Review Questions

1. Where might flash memory be found on a system board?
2. Name two ways that a SIMM and a DIMM are alike. Name two ways they are different.
3. How many pins are on a DIMM? What are the two possible numbers of pins on a SIMM?
4. Which is faster, EDO memory or BEDO memory?
5. How does a memory cache speed up computer processing?
6. Explain the difference between a level 1 cache and a level 2 cache.
7. What types of memory can you use for a 100-MHz system board?
8. If your system board supports FPM memory, will EDO memory still work on the board?
9. Looking at a SDRAM DIMM, how can you know for certain the voltage needed by the module?
10. If your system board supports ECC SDRAM memory, can you substitute SDRAM memory that does not support ECC? If your system board supports buffered SDRAM memory, can you substitute unbuffered SDRAM modules?
11. What might be a symptom in Windows of unreliable memory on a system board?
12. If your system board calls for 60 ns memory, can you substitute 70 ns memory? Why, or why not?
13. When buying memory, what can you look for that might indicate that the memory is remanufactured?
14. What are the two major categories of static RAM memory?
15. When might there not be any SRAM on a system board?
16. Which operating system operates totally in real mode? Which OS operates totally in protected mode? Which OS can switch back and forth between the two?
17. Explain the difference between the terms memory and memory address.
18. What is memory mapping? When does it occur?
19. What area of memory does video BIOS use?
20. When might you configure Windows to provide emulated expanded memory?
21. Contrast virtual memory and a RAM drive.
22. How are the two commands DEVICEHIGH and LOADHIGH the same? How are they different?
23. Which is executed first in the boot process: CONFIG.SYS or AUTOEXEC.BAT?
24. When virtual memory is stored on a hard drive, what is it called?
25. Why do you want to use 32-bit drivers in Windows 9x rather than 16-bit drivers?

26. On a Pentium system board, why must you use SIMMs in pairs?
27. If the density of a 72-pin SIMM is 4 × 32, what is the size of the SIMM?
28. Using your home or lab computer, answer these questions:
    a. How much RAM is on the system board?
    b. How much virtual memory is used?
    c. Is the virtual memory permanent or temporary?
29. When installing a RIMM memory module, what must be installed in each empty RIMM slot? Why?
30. After installing new memory in a PC, how does the system know that the new memory is present?

PROJECTS

Unless you follow proper procedures, working inside your computer can cause serious damage to both you and your computer. To ensure safety in your work setting, follow every precaution listed in the *Read This Before You Begin* section following this book's Introduction.

Using Nuts & Bolts to Examine Memory

Use the Nuts & Bolts utility, Discover Pro, to answer the following questions about the memory on your PC. Using Windows 9x, click **Start**, point to **Programs**, **Nuts & Bolts**, and select **Discover Pro** from the list of Nuts & Bolts utilities. From Discover Pro, select **System** and then **Advanced**. Select the **BIOS/CMOS** icon in the Advanced section.

1. What memory addresses are currently being used by your computer to hold System BIOS?
2. Are any System BIOS programs being shadowed into RAM?
3. What is the memory address of the I/O address table where the keyboard buffer is located?
4. Convert this keyboard buffer address to decimal (see Appendix D).
5. Click **Summary** to return to opening screen.

From Discover Pro, select **Memory** and then **Advanced** to answer these questions.

6. What memory addresses are currently assigned to video?
7. What memory addresses in upper memory are available for UMBs?

Help Desk Support

1. A friend calls who is sitting at his computer and asks you to help him determine how much RAM he has on his system board. Step him through the process. List at least two different ways to find the answer. He is using DOS 6.22.
2. Answer Question 1 above, assuming that your friend is using Windows 98.
3. A customer calls who has Norton Utilities on her PC. She has an applications software program installed, and she wants to know whether it is a 16-bit or 32-bit application. Unfortunately, she does not have the documentation for the software available. Help her to find the answer to her question.

Planning and Pricing Memory

Read the documentation and look at the system board of your computer. What is the maximum amount of memory the banks on your system board can accommodate? How much memory do they now hold? Look in a computer catalog, such as *Computer Shopper*, and determine how much it costs to fill the banks to full capacity. Don't forget to match the speed of the modules already installed, and plan to use only the size modules your computer can accommodate.

Using Upper Memory

If you have access to a computer running DOS, does this computer make the best use of upper memory? Get a printout of the files CONFIG.SYS and AUTOEXEC.BAT and the results of the MEM/C command. Study the printouts to determine any improvements that could free more conventional memory for applications.

Installing a TSR

DOSKEY is a very handy TSR. When it is installed, you can retrieve previously issued DOS commands by pressing the Up Arrow key. See your DOS manual for other uses of DOSKEY. To install DOSKEY at the C prompt, use this command:

```
C:\>DOSKEY
```

To install DOSKEY so that it is available every time your computer boots up, you must include the command in your AUTOEXEC.BAT file. Installing DOSKEY in upper memory so that it does not use any conventional memory requires that you create upper memory blocks. List the commands in CONFIG.SYS and AUTOEXEC.BAT to install DOSKEY in upper memory.

Upgrading Memory

To practice installing additional memory in a computer in a classroom environment, remove the SIMMs or DIMMs from one computer and place them in another computer. Boot the second computer and check that it counts up the additional memory. When finished, return the borrowed modules to the original computer.

Using MSD to View Memory

Access MSD, and then access its view of memory on your computer. Get a printout of the MSD memory report.

How Memory is Allocated

Use the /M option on the MEM command to list the ways in which three TSRs currently installed in the memory of your computer are using this memory. For example, for a mouse driver named MOUSE.SYS currently installed, type this command:

```
MEM/M MOUSE
```

Using MemMaker

On a PC using DOS and Windows 3.x that does not provide access to upper memory, print a MEM report similar to Figure 4-17. Use MemMaker to create the commands necessary at bootup to manage memory effectively. Print a new MEM report to verify the improvement. Print copies of the AUTOEXEC.BAT and CONFIG.SYS files before and after using MemMaker.

Troubleshooting Memory

Loosen a memory module on a system board and boot the PC. What error do you get? Don't forget to follow rules to protect the PC against ESD as you work.

Plan Memory Installation

Fill out the following table, referring back to Table 4-5 in the text.

Memory Configurations for a Pentium System Board

Bank 0	Bank 1	Total Memory
16 MB	16 MB	64 MB
16 MB	32 MB	
32 MB		64 MB
	4 MB	72 MB
32 MB	8 MB	
32 MB	16 MB	
	32 MB	128 MB
0		8 MB
0	8 MB	
	16 MB	32 MB
0		64 MB

Assume that this system board is using EDO, 60-ns, 72-pin SIMMs and currently has 32 MB of memory in Bank 1. Find and make a copy of a memory module ad. Mark in the ad the memory module that is needed by this system board to upgrade to 64 MB. How many modules are needed, and what is the total cost?

Real Mode in Windows 9x

Create a shortcut on your desktop to provide a DOS prompt running in real mode. *Hint*: Create a shortcut for COMMAND.COM and then change the properties for this shortcut. List the series of events that happen when you execute the DOS prompt shortcut and then return to regular Windows 9x.